

(19) World Intellectual Property Organization
International Bureau



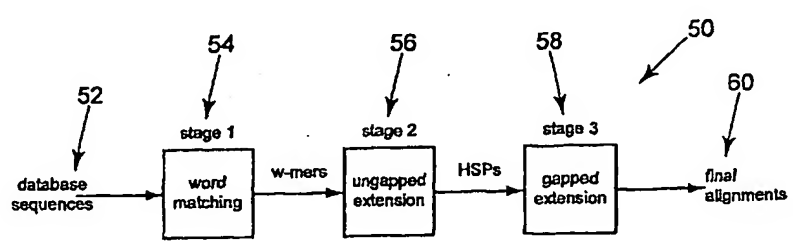
(43) International Publication Date
14 September 2006 (14.09.2006)

PCT

(10) International Publication Number
WO 2006/096324 A2

- (51) International Patent Classification: Not classified Joseph Marion [US/US]; 6260 Cates Avenue, #2E, St. Louis, MO 63130 (US).
- (21) International Application Number: PCT/US2006/006105 (74) Agents: KERCHER, Kevin M. et al; Thompson Coburn LLP, One US Bank Plaza, St. Louis, Missouri 63101 (US).
- (22) International Filing Date: 22 February 2006 (22.02.2006) (81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
60/658,418 3 March 2005 (03.03.2005) US
60/736,081 11 November 2005 (11.11.2005) US
- (71) Applicant (for all designated States except US): WASHINGTON UNIVERSITY [US/US]; One Brookings Drive, St. Louis, MO 63130 (US).
- (72) Inventors; and
- (75) Inventors/Applicants (for US only): BUHLER, Jeremy Daniel [US/US]; 551 Donne Avenue, St. Louis, MO 63130 (US). CHAMBERLAIN, Roger Dean [US/US]; 64 Notre Dame Drive, St. Louis, MO 63141 (US). FRANKLIN, Mark Allen [US/US]; 7456 Stratford Avenue, St. Louis, MO 63130 (US). GYANG, Kwame [GH/US]; 4444 West Pine Boulevard, #114, St. Louis, MO 63108 (US). JACOB, Arpith Chacko [IN/US]; 6012 McPherson Avenue, Apt. 2E, St. Louis, MO 63112 (US). KRISHNAMURTHY, Praveen [IN/US]; 6429 Cates Avenue, #2E, St. Louis, MO 63130 (US). LANCASTER,
- (84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).
- Published:
— without international search report and to be republished upon receipt of that report
- For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND APPARATUS FOR PERFORMING BIOSEQUENCE SIMILARITY SEARCHING



(57) Abstract: A system and method for performing biological sequence similarity searching is disclosed. This includes a programmable logic device configured to include a pipeline that comprises a matching stage, the matching stage being configured to receive a data stream comprising a plurality of possible matches between a plurality of biological sequence data strings and a plurality of substrings of a query string. The pipeline may further include a ungapped extension prefilter stage located downstream from the matching stage, the prefilter stage being configured to shift through pattern matches between the biological sequence data strings and the plurality of substrings of a query string and provide a score so that only pattern matches that exceed a user defined score will pass downstream from the prefilter stage. The matching stage may include at least one Bloom filter.

WO 2006/096324 A2

METHOD AND APPARATUS FOR PERFORMING BIOSEQUENCE SIMILARITY SEARCHING

CROSS-REFERENCE TO RELATED APPLICATIONS

- [0001] This application claims the benefit of U.S. Provisional Application No. 60/658,418, filed on March 3, 2005 and claims the benefit of U.S. Provisional Application No. 60/736,081, filed on November 11, 2005.

FIELD OF THE INVENTION

- [0002] The present invention relates to the field of biosequence similarity searching. In particular, the present invention relates to the field of searching large databases of biological sequences for strings that match a query sequence.

BACKGROUND OF THE INVENTION

- [0003] The amount of biosequence data being produced each year is growing exponentially. Extracting useful information from this massive amount of data efficiently is becoming an increasingly difficult task. The databases of genomic DNA and protein sequences are an essential resource for modern molecular biology. This is where a computational search of these databases can show that a DNA sequence acquired in the lab is similar to other sequences of a known biological function, revealing both its role in the cell and its history over evolutionary time. A decade of improvement in DNA sequencing technology has driven exponential growth of biosequence databases such as NCBI GenBank, which has doubled in size every twelve (12) to sixteen (16) months for the last decade and now stands at over forty-five (45) billion characters. These technological gains have also generated more novel sequences, including entire mammalian genomes, which keep search engines very busy.

- [0004] Examples of this type of searching can be found in a paper entitled *Biosequence Similarity Search On The Mercury System*, P. Krishnamurthy, J. Buhler, R. D. Chamberlain, M. A. Franklin, K. Gyang, and J. Lancaster, In Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP04), September 2004, Pages 365-375 (2004). Another example is a paper entitled: *NCBI BLASTN STAGE 1 INRECONFIGURABLE HARDWARE*, by Kwame Gyang, Department of Computer Science and Engineering, Washington University, August 2004, Technical Report WUCSE-2005-30. Another example is a paper entitled "*BLASTN Redundancy Filter in Reprogrammable Hardware*", by C. Behrens, J. Lancaster, and B. Wun, Department of

Computer Science and Engineering, Washington University, Final Project Submission, Fall 2003. These papers are each incorporated by reference, in their entirety.

[0005] There is a growing need for a fast and cost effective mechanism for extracting useful information from biosequence data.

SUMMARY OF INVENTION

[0006] In one aspect of this invention, a digital logic circuit for performing biological sequence similarity searching is disclosed. This digital logic circuit includes a programmable logic device configured to include a pipeline comprising a Bloom filter stage, the Bloom filter stage being configured to receive a data stream comprising a plurality of biological sequence data strings and filter the biological sequence data stream to identify a plurality of possible matches between the sequence data strings and a plurality of substrings of a query string.

[0007] In another aspect of this invention, a system for performing seeded alignment searching is disclosed. This system includes reconfigurable hardware configured to implement a seeded alignment searching pipeline, the pipeline comprising a Bloom filter stage, the Bloom filter stage being configured to receive a data stream, the data stream comprising a plurality of data strings and filter the data stream to identify a plurality of possible matches between the stream data strings and a plurality of substrings of a query string.

[0008] In still another aspect of this invention, a method for performing biological sequence similarity searching is disclosed. This method includes processing a biological sequence data stream through a Bloom filter stage implemented on a programmable logic device, the biological sequence data stream comprising a plurality of biological sequence data strings, the Bloom filter stage comprising a Bloom filter that is programmed with a plurality of substrings of a query string and configured to identify a plurality of possible matches between the biological sequence data strings and the query substrings.

[0009] In yet another aspect of this invention, a digital logic circuit for performing biological sequence similarity searching is disclosed. This digital logic circuit includes a programmable logic device configured to include a pipeline that comprises a matching stage, the matching stage being configured to receive a data stream comprising a plurality of possible matches between a plurality of biological sequence data strings and a plurality of substrings of a query string and the pipeline further comprises a prefilter stage located downstream from the matching stage, the prefilter stage being configured to shift through pattern matches between

the biological sequence data strings and the plurality of substrings of a query string and provide a score so that only pattern matches that exceed a user defined score will pass downstream from the prefilter stage.

[0010] In still yet another aspect of this invention, a method for performing biological sequence similarity searching is disclosed. This method includes processing a biological sequence data stream with a programmable logic device configured to include a pipeline that comprises a matching stage, the matching stage being configured to receive a data stream comprising a plurality of possible matches between a plurality of biological sequence data strings and a plurality of substrings of a query string and utilizing a prefilter stage located downstream from the matching stage in the pipeline, the prefilter stage being configured to shift through pattern matches between the biological sequence data strings and the plurality of substrings of a query string and provide a score so that only pattern matches that exceed a user defined score pass will downstream from the prefilter stage.

[0011] These are merely some of the innumerable aspects of the present invention and should not be deemed an all-inclusive listing of the innumerable aspects associated with the present invention.

BRIEF DESCRIPTION OF DRAWINGS

[0012] For a better understanding of the present invention, reference may be made to the accompanying drawings in which:

[0013] Figure 1 provides a block diagram overview of a three (3) stage pipeline in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention broken down into three (3) main stages;

[0014] Figure 2 provides an exemplary block diagram of a Stage 1 pipeline, e.g., BLASTN, in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;

[0015] Figure 3 is a general schematic of a hardware platform for a biological data pipeline in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;

[0016] Figure 4 depicts a sample portion of a biological sequence data stream;

[0017] Figure 5 depicts a query sequence can be broken down into multiple w-mers in connection with the database sequence shown in Figure 3;

[0018] Figure 6(a) depicts an exemplary Bloom filter stream in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;

- [0019] Figure 6(b) shows an arrangement of Bloom filters configured to share the same dual port block random access memories (BRAMs) in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0020] Figure 6(c) depicts a Bloom filter stage that comprises sixteen (16) Bloom filters formed from four (4) Bloom filter arrangements, as shown in Figure 6(b), in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0021] Figure 7 is a schematic and flow chart of control logic that is either within or associated with a Bloom filter stage and readily configured to identify the starting position within the database sequence of the database w-mer that is a possible match with a query w-mer;
- [0022] Figure 8 illustrates an exemplary hashing stage and hash table in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0023] Figure 9 depicts an exemplary data window associated with the analysis of ungapped extension with analysis in two directions within a predefined window of data;
- [0024] Figure 10 depicts an exemplary data window associated with the analysis of ungapped extension with analysis in a single direction within a predefined data window in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0025] Figure 11 is a block diagram overview of the three (3) stage pipeline of Figure 1, wherein the first and second stages are each broken down into two (2) substages;
- [0026] Figure 11A is a block diagram overview of the three (3) stage pipeline of an alternative embodiment of the present invention shown in Figure 1, wherein only the first stage is broken down into two (2) substages;
- [0027] Figure 12 is an illustrative, but nonlimiting, listing of pseudocode utilized in an ungapped extension algorithm in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0028] Figure 13 is an illustrative, but nonlimiting, block diagram of a prefilter stage for an ungapped extension in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0029] Figure 13A is an illustrative, but nonlimiting, block diagram of a window lookup module of the prefilter stage for an ungapped extension, as shown in Figure 13;
- [0030] Figure 14 is an illustrative, but nonlimiting, block diagram of a base comparator, as shown in Figure 13, in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;

- [0031] Figure 15 is an illustrative, but nonlimiting, block diagram of an array of scoring stages, as shown in Figure 13, in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0032] Figure 16 is an illustrative, but nonlimiting, block diagram of a single two step scoring stage, as shown in Figure 15, in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0033] Figure 17 is a graphical representation of throughput of an overall pipeline as a function of ungapped extension throughput for queries of sizes 20 kbases and 25 kbases;
- [0034] Figure 18 is a graphical representation of speedup of an overall pipeline as a function of ungapped extension throughput for queries of sizes 20 kbases and 25 kbases with prefiltering in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention;
- [0035] Figure 19 is a table of sensitivity results for an illustrative, but nonlimiting, prefilter for the ungapped extension in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention; and
- [0036] Figure 20 is a flow chart of the algorithm shown in Figure 12 utilized in an ungapped extension in accordance with a preferred, nonlimiting, illustrative embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

- [0037] In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the invention. However, it will be understood by those skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known methods, procedures, and components have not been described in detail so as to obscure the present invention.
- [0038] In an effort to improve upon the speed of biological similarity searching, the inventors herein have developed a novel and unique method and system whereby Stage 1 of the software pipeline known in the art as BLAST (Basic Local Alignment Search Tool), and more particularly the BLAST application BLASTN, is implemented via programmable logic device, e.g., reconfigurable hardware, such as, but not limited to, FPGAs (field programmable gate arrays). In addition, this present invention decides in Stage 2 of the software pipeline, ungapped extension, whether each w-mer emitted from word matching is worth inspecting by the more computationally intensive, gapped extension by shifting through pattern matches between query and database. It is important to identify and discard

spurious w-mers as early as possible in the BLAST pipeline because later stages are increasingly more complex.

[0039] BLAST compares a query sequence to a biosequence database to find other sequences that differ from it by a small number of edits (single-character insertions, deletions, or substitutions). Because direct measurement of the edit distance between sequences is computationally expensive, BLAST uses a variety of heuristics to identify small portions of a large database that are worth comparing carefully to the biosequence query.

[0040] BLAST includes a pipeline of computations that filter a stream of characters from a database to identify meaningful matches to a query. To keep pace with growing databases and queries, this stream must be filtered at increasingly higher rates. One path to higher performance is to develop a specialized processor that offloads part of BLAST'S computation from a general-purpose processor, e.g., CPU. Illustrative, but nonlimiting, examples of processors that are known to accelerate or replace BLAST include the ASIC-based Paracel GeneMatcher™ and the FPGA-based TimeLogic DecypherBLAST™ engine.

[0041] There is also a recently developed new design for an accelerator, i.e., the FPGA-based MERCURY BLAST engine. This MERCURY BLAST engine utilizes fine-grained parallelism in BLAST'S algorithms and the high input/output (I/O) bandwidth of current commodity computing systems to deliver a speedup that is at least one (1) to two (2) orders of magnitude over the software BLAST that can be deployed on a standard processor located in a laboratory.

[0042] The MERCURY BLAST engine is a multistage pipeline and this present invention involves word matching, i.e., a first stage (Stage 1), and an ungapped extension stage, i.e., a second stage (Stage 2). The present invention includes a prefilter stage between the word matching stage and the ungapped extension stage that sifts through pattern matches between a query sequence and a biological database sequence and decides whether to perform a more accurate, but computationally expensive, comparison between them. The present invention also provides a more fruitful acceleration of variable-length string matching that is robust to character substitutions. The illustrative, but nonlimiting, preferred implementation is compact, runs at high clock rates, and can process one pattern match for every clock cycle.

[0043] Figure 1 shows a block diagram of a high level overview of a three (3) stage pipeline that is generally indicated by numeral 50. The database sequences 52 are shown entering the pipeline 50. The first stage (Stage 1) 54 provides a word matching function that detects substrings of fixed length "w" in the stream that perfectly match a substring of the query, e.g., w = 11, for DNA. These short matches are also known as "w-mers." Each matching w-mer

is forwarded to a second stage (Stage 2), which is an ungapped extension 56, which extends the w-mer on either side to identify a longer pair of sequences around the w-mer that match with at most a small number of mismatched characters. These longer matches are identified as high-scoring segment pairs ("HSP"s) or ungapped alignments.

[0044] Finally, every high-scoring segment pair (HSP) that has both enough matches and sufficiently few mismatches is passed to the third stage (Stage 3), which is gapped extension 58, which preferably uses the Smith-Waterman dynamic programming algorithm to extend the high-scoring segment pair (HSP) into a gapped alignment. Gapped alignment is a pair of similar regions that may differ by only a few arbitrary edits. The software, e.g., BLAST, reports only final gapped alignments with many matches and few edits 60.

[0045] Research has determined that with the standard NCBI BLASTN software program, 83.9% of the computational time was spent in the first stage (Stage 1) 54 and 15.9% of the time was spent in the second stage (Stage 2) 56. Therefore, to achieve a significant speedup, e.g., greater than six (6) times, of BLASTN would require acceleration of both the first stage (Stage 1) 54 and the second stage (Stage 2) 56. NCBI is the National Center for Biotechnology Information, which is a multi-disciplinary research group, composed of computer scientists, molecular biologists, mathematicians, biochemists, research physicians, and structural biologists concentrating on basic and applied research in computational molecular biology utilizing a variety of software databases and tools.

[0046] A block diagram overview of the first stage (Stage 1) 54, referenced in Figure 1, is shown in Figure 2, which is a pipeline 100, e.g., BLASTN Stage 1, in accordance with a preferred embodiment of the present invention. The input processing unit 104, the Bloom filter stage 106, the string buffer unit 108, the hashing stage 110, the hash (look-up) table 112, and the redundancy filter 114 are preferably implemented on a programmable logic device. An illustrative, but nonlimiting, example of a programmable logic device 102 can include a field programmable gate array (FPGA). A hash table 112 is preferably implemented on a memory unit, e.g., static random access memory (SRAM) 118, which can be off-chip from the programmable logic device 102.

[0047] The placement of the hash table 112 as being off-chip is optional and is only a result of the memory constraints for field programmable gate arrays (FPGAs), wherein readily available FPGAs do not have sufficient capacity to store the hash table and provide the functionality of the various pipeline stages. However, if capacity for a field programmable gate array (FPGA) sufficiently increases in the future, the hash table 112 can also be implemented on an FPGA.

[0048] A preferred hardware platform for pipeline 100 is that disclosed in Figure 3 and is generally indicated by numeral 800. Additional details about this hardware platform are disclosed in: (1) U.S. Patent No. 6,711,558 entitled "Associative Database Scanning and Information Retrieval"; (2) pending U.S. patent application 10/153,151, filed May 21, 2002 and entitled "Associative Database Scanning and Information Retrieval Using FPGA Devices" (published as U.S. patent application publication 2003/0018630); (3) pending PCT application PCT/US04/16398 filed May 21, 2004 and entitled "Intelligent Storage and Processing Using FPGA Devices"; and (4) a paper on *"Achieving Real Data Throughput for an FPGA Co-Processor on Commodity Server Platforms"* by Chamberlain et al, published in Proc. of 1st Workshop on Building Block Engine Architectures for Computers and Networks, Boston, MA, October 2004; and a paper on *"Acceleration of Ungapped Extension in Mercury BLAST"* by Lancaster et al., was published on November 12, 2005 for the Seventh (7th) Workshop on Media and Streaming Processors held in conjunction with the Thirty-Eighth (38th) International Symposium on Microarchitecture (MICRO-38) in Barcelona, Spain, the entire disclosures of all of which are incorporated herein by reference.

[0049] The input processing unit 104 is preferably configured to receive a biological sequence data stream 120 as an input from a sequence database, which is shown in Figure 2. This database stream 120 represents a database sequence that can be broken down into a plurality of substrings, each substring having a base length "w." Such substrings are referred to herein as database w-mers. The input processing unit 104 preferably outputs a parallel stream of q database w-mers (or sequence w-mers) 122i, wherein q represents the total number of database w-mers that can be simultaneously processed by the Bloom filter stage 106.

[0050] A depiction of a sample portion of a biological sequence data stream is indicated by numeral 120, which is shown in Figure 4. Stream 120 comprises a sequence of bases (in this example, A, C, G, or T, which correspond to the bases of a DNA sequence). In this example, each base can be represented by 2 or 4 bits, as is known in the art. If w equals 11 and q equals 8, this stream 120 can be converted by the input processing unit 104 into 8 parallel w-mers as shown in Figure 5.

[0051] The Bloom filter stage 106, as shown in Figure 2, operates to process each database w-mer to determine whether it is a possible match with a query w-mer. A query w-mer being a substring of base length "w" derived from a query sequence. By definition, Bloom filters will not produce any false negatives between the database w-mers and the query w-mers, but Bloom filters are highly likely to produce some number of false positive matches between the

database w-mers and the query w-mers. Because of the presence of these false positives, the "yes" responses of the Bloom filter to database w-mers are best referred to as possible matches.

[0052] Before processing database w-mers, the Bloom filters within Bloom filter stage 106 will need to be programmed/keyed with query w-mers from a query sequence (or query string). The query sequence is some string of bases for which a person wants to compare with the biological database sequence to find matches. The query sequence can be broken down into multiple w-mers as shown in Figure 5 in connection with the database sequence. For a given search, the value of w is the same for both the query w-mers and database w-mers. Each Bloom filter within Bloom filter stage 106 is preferably programmed with all of the query w-mers developed from the query sequence. In an illustrative, but nonlimiting, preferred embodiment, the Bloom filter stage 106 comprises a plurality of parallel Bloom filters. Also, it is preferred, but not necessary, that the parallel Bloom filters are copies of one another. A Bloom filter is said to be in parallel with another Bloom filter when both of those Bloom filters are configured to simultaneously process w-mers. Programming of the Bloom filter stage 106 can be achieved via appropriate control commands asserted by the input processing unit 104 (in response to control input to the pipeline 100).

[0053] A preferred Bloom filter architecture within the Bloom filter stage 106 is shown in Figures 6(a)-6(c). Figure 6(a) depicts an exemplary Bloom filter 400. Within the Bloom filter 400, a database w-mer is processed by "k" hash functions 402, each hash function "f" operating to map the database w-mer to a bit in the 1 x m dual port block RAM (BRAM) unit 404i that corresponds to that hash function i. If all of the bits to which that database w-mer maps within BRAMs 4041 through 404k are set (e.g., equal to 1), then logic unit 406, e.g., AND logic unit, will receive all 1's on its input lines, thereby producing a "yes" response as to whether that database w-mer is a possible match with a query w-mer.

[0054] The bits within the 1 x m dual port BRAMs 404 are set if during the Bloom filter programming process, a query w-mer maps to that bit location. Because the Bloom filter is programmed with all of the query w-mers from the query sequence, the output from logic unit 406, e.g., AND logic unit, will never produce a false negative, but may produce a false positive.

[0055] The hash functions 402 are preferably the same hash functions selected from the H3 family of hash functions as used in the hashing stage 110. In the illustrative, but nonlimiting, example of Figure 6(a), the 1 x m memory units 404 to which the hash functions 402 map can include dual-ported block random access memories (BRAMs). However, by virtue of being

dual ported, multiple Bloom filters can share access to the same BRAM 404, thereby providing a substantial savings in the resources consumed on the programmable logic device 102, e.g., FPGA, when implementing the Bloom filter stage 106.

[0056] A description of the operation of Bloom filters can be found in U.S. Patent Application No. 10/640,513, which was published as U.S. Patent Application No. 2005/0086520, and was filed on August 14, 2003, which is incorporated herein by reference in its entirety.

[0057] As an additional realization of consumed resources, the dual port BRAMs 404 are preferably double clocked to allow four Bloom filters 106 to share the same dual port BRAM 404. Thus, during a given clock cycle, 4 accesses can be made to each dual port BRAM 404i; thereby allowing each dual port BRAM to support four Bloom filters 106 each clock cycle.

[0058] An arrangement 410 of four (4) Bloom filters 106 configured to share the same k BRAMs 404i through 404k is shown in Figure 6(b). Further still, if it is feasible to triple clock the BRAMs 404, it is worth noting that each dual port BRAM 404 could support six (6) Bloom filters.

[0059] A Bloom filter stage 106 that includes at least sixteen (16) Bloom filters is shown in Figure 6(c) formed from four (4) Bloom filter arrangements 410 that are shown in Figure 6(b). Such a Bloom filter stage 106 is capable of simultaneously processing a different database w-mer through each parallel Bloom filter. As such, the Bloom filter stage 106 can determine whether possible matches exist for multiple (sixteen (16) in a preferred, but nonlimiting, embodiment) database w-mers simultaneously. While the Bloom filter stage 106 comprises sixteen (16) parallel Bloom filters in a preferred embodiment, it should be noted that more or less parallel Bloom filters can be used in the practice of the present invention, as would be understood by a person having ordinary skill in the art following the teachings presented herein.

[0060] The control logic that is either within Bloom filter stage 106 or associated with the Bloom filter stage 106 can be readily configured to identify the starting position within the database sequence of the database w-mer that is a possible match with a query w-mer. An illustrative, but nonlimiting, example of how this control logic could be configured with an illustrative flow chart diagram is shown in Figure 7 and is generally indicated by numeral 900. The first step is for a counter to be updated each time a new clock cycle is received to reflect the new starting point within the database sequence of w-mer for that clock cycle, as indicated by numeral 902. Depending on which logic unit 406i (or units), e.g., AND logic

unit, finds a possible match 904, an appropriate starting position for a possibly matching w-mer can be recorded. If a possible match is found, then i can be recorded 906 and for each recorded value of i , return a starting point (SP) plus the value of i as the pointer for a new starting position for a possible match in the database sequence 908. The process then determines if there is a new clock cycle 910. If the answer is negative, then this determination will be repeated until there is a positive determination and the starting point (SP) can be increased by q indicated by process step 912, wherein q represents the total number of database w-mers that can be simultaneously processed by the Bloom filter stage 106. The process then returns to step 904 to determine if a possible match can be found.

[0061] If no match is found in process step 904, the process then determines if there is a new clock cycle 910. If the answer is negative, then this determination will be repeated until there is a positive determination and the starting point (SP) can be increased by q indicated by process step 912, wherein q represents the total number of database w-mers that can be simultaneously processed by the Bloom filter stage 106. The process then returns to step 904 to determine if a possible match can be found.

[0062] Each of these starting positions for a possible match is preferably then passed on to the string buffer unit 108, as shown in Figure 2, as an output from the Bloom filter stage 106. Optionally, an output 124i from the Bloom filter stage can comprise the actual database w-mer that triggered the possible match, either in addition to or instead of the starting position pointer.

[0063] A preferred range of values for w is 5 to 15, with 11 being the most preferred value. However, it should be noted that values outside this range can also be used if desired by a practitioner of the invention. A preferred value for q is 16. A preferred value for k is 6. Preferred values for m are 32 kbits or 64 kbits. However, other values for q , k , and m can be used if desired by a practitioner of the invention. Additional details about the preferred Bloom filter stage 106 as well as alternative embodiments are described in publications that were previously incorporated by reference.

[0064] The string buffer unit 108 operates to receive the parallel outputs 124i through 124q that comprise pointers that identify the starting position(s) in the database sequence for each database w-mer that is a possible match with a query w-mer and/or the possibly matching database w-mers themselves. String buffer unit 108 produces as an output 126 a serialized stream of pointers to these starting positions and/or possibly matching database w-mers. Additional details about the preferred string buffer unit 108 are described in publications that were previously incorporated by reference.

[0065] For each received database w-mer pointer or database w-mer within stream 126, hashing stage 110 operates to confirm that a possible match exists between a database w-mer and a query w-mer and determine a pointer to the query w-mer that may possibly match the database w-mer and is shown in Figure 2. If stream 126 includes only pointers and not the database w-mers themselves, then it is preferred that hashing stage 110 also receive stream 120 so that the hashing stage can extract the appropriate possibly matching database w-mers from stream 120 using the starting position pointers in stream 126. Hash functions within the hash stage operate to map each possibly matching database w-mer to a position in hash table 112.

[0066] An exemplary hashing stage 110 and hash table 112 in accordance with a preferred embodiment of the present invention is shown in Figure 8. The hash table 112 preferably comprises a primary table 602, a secondary table 604, and a duplicate table 606. Stored at each address in primary table 602 is preferably an entry that includes a pointer to the starting position in the query sequence for the query w-mer that was mapped to that address by the hash functions for hashing stage 110. For any query w-mers that are duplicated at multiple locations within the query sequence, a duplicate bit in that query w-mer's corresponding entry in primary table 602 is preferably set.

[0067] To build the hash table 112, hash functions are used to map each query w-mer to an address in the hashing stage 110. The creation of these hash functions is preferably performed by software prior to streaming the database w-mers through the pipeline 100. Once the hash functions are created by software, these hash functions are preferably loaded into hashing stage 110 during the programming process prior to streaming the database w-mers through the pipeline. As new query biological sequences are used to program the hashing stage 110, there is a displacement table utilized by the hashing functions that is preferably updated to reflect the new query w-mers. The hashing functions themselves can remain the same. Thereafter, the hash table 112 can be programmed by processing the query w-mers through the hashing stage 110 while an appropriate control command is asserted. If desired, ordinary hash functions can be used to build the hash table 112 and process database w-mers in the hashing stage 110. However, because ordinary hashing typically produces multiple collisions, and because the hash table 112 can be stored off the programmable logic device 102, e.g., FPGA, in memory 118, e.g., SRAM, it is desirable use hashing techniques other than ordinary hashing to minimize the number of memory look-ups, and thereby avoid a potential pipeline bottleneck. In the hashing process, a collision occurs if when building a hash table 112, two different query w-mers map to the same address in the primary table 602.

With ordinary hashing, such collisions are not uncommon. To resolve such collisions, a secondary table 604 is preferably used that provides different addresses for the different w-mers that mapped to the same address in the primary table 602, as is well-known. A collision bit in the address of the primary table 602 where the collision occurred is then set to identify the need to access the secondary table to properly map that w-mer.

[0068] A secondary set of hash functions are preferably used by the hashing stage 110 to map database w-mers to the secondary table 604. This secondary set of hash functions can be either ordinary hash functions, perfect hash functions, or near perfect hash functions, depending upon the preferences of a practitioner of the present invention. In a preferred hashing operation, the hash functions that map database w-mers to the primary table 602 and the hash functions that map database w-mers to the secondary table 604 simultaneously process all received database w-mers. If the address in the primary table 602 to which the database w-mer maps has its collision bit set, then the hashing stage logic uses the address in the secondary table 604 to which that database w-mer mapped, thereby avoiding the need for the hashing stage to re-hash a database w-mer if that database w-mer maps to an address where a collision occurred.

[0069] In a first alternative embodiment that seeks to minimize the number of hash table look-ups, perfect hashing functions are used to build the hash table 112 and perform the mapping of hashing stage 110. If perfect hashing is used to build hash table 112, then the secondary table 604 is not needed. Perfect hashing functions are explained in greater detail in some of the publications incorporated by reference.

[0070] In a second alternative embodiment, there is a need to reduce the number of hash table look-ups so that near perfect hashing functions are used to build the hash table 112 and perform the mapping of hashing stage 110. Near perfect hashing can be defined as hashing that operates to provide a decreased likelihood of collisions relative to the likelihood of collisions provided by ordinary hashing functions, and wherein the hashing approaches a perfect hash the longer that it executes. Near perfect hashing functions are explained in greater detail in some of the publications incorporated by reference. The preferred near perfect hashing functions are those in the family H3 chosen to be of full rank. The family of hashing functions that is designated as "H3" as well as the concept of "full rank" are fully described and disclosed in numerous publications in this area. More generally, the near-perfect hashing algorithm described herein may be used to generate a mapping of a set of binary strings of common length equal to n, referred to as keys, to indices into a hash table, so that few or no pairs of keys map to the same location in the table. The hashing logic of

hashing stage 110 is preferably implemented on the programmable logic device 102, e.g., FPGA, to map possibly matching database w-mers to their respective positions in the query sequence.

[0071] If the hashing stage 110 maps a possibly matching database w-mer to an empty entry in the hash table 112, then that database w-mer can be discarded as a false positive. If the hashing stage 110 maps a possibly matching database w-mer to an entry in the hash table 112 that contains a pointer to a position in the query sequence, then neither the duplicate bit nor the collision bit will be set. Then that pointer points to the only starting position in the query sequence for the query w-mer that is a possible match to that database w-mer.

[0072] If the hashing stage 110 maps a possibly matching database w-mer to an entry in the hash table 112 for which the collision bit is set, then the hashing stage 110 looks to the address in the secondary table 604 where the secondary hash function mapped that possibly matching database w-mer. Once the proper entry in the secondary table 604 for the database w-mer is identified, then the secondary table entry is processed in the same manner as entries in the primary table (whose collision bits are not set).

[0073] If the hashing stage 110 maps a possibly matching database w-mer to an entry in the hash table 112 that contains a pointer to a position in the query sequence, and the duplicate bit is set, then that pointer points to one of a plurality of starting positions in the query sequence for the query w-mer that is a possible match to that database w-mer. To obtain the starting position(s) for each duplicate query w-mer, the duplicate table 606 is then preferably accessed. Also included in each primary table entry whose duplicate bit is set is a pointer to an address in the duplicate table 606 for that query w-mer. At that address, the duplicate table address preferably includes an entry that identifies the total number of duplicate query w-mers that exist in the duplicate table 606 for a given query w-mer. This total number is preferably followed by a contiguous array comprising, for each w-mer that is a duplicate of the query w-mer at issue, a pointer to the starting position in the query sequence for that duplicate query w-mer. Alternatively, linked list techniques can be used to chain the duplicate w-mer pointers together. Further still, the system can be configured to cancel a query sequence search if, during programming of the hash table 110, the number of duplicate w-mers in the query sequence is sufficiently large to exceed the capacity of the duplicate table 606.

[0074] Therefore, the hashing stage 110 operates to identify for each database w-mer that is a possible match to a query w-mer, the starting position in the query sequence for each query w-mer that is a possible match thereto. The preferred output 128 from the hashing stage 110

is a stream of ordered pairs, where each ordered pair comprises a pointer to the starting position in the database sequence of the database w-mer that is a possible match with a query w-mer and a pointer to a starting position in the query sequence of the query w-mer that possibly matches the database w-mer. If desired, comparison logic (not shown) can be optionally implemented on the programmable logic device 102, e.g., FPGA, following the hashing stage that operates to compare the database w-mer and query w-mer for each ordered pair to eliminate false positives.

[0075] It is worth noting that the role of the duplicate table 606 can be removed from the hash table 112 and placed either in separate memory, e.g., SRAM, or on the programmable logic device 102, e.g., FPGA, as its own pipeline stage downstream from the hashing stage 110 if desired by a practitioner of the invention.

[0076] The redundancy filter stage 114 is preferably configured to remove ordered pairs from stream 128 that are deemed redundant with other ordered pairs within stream 128. The database-to-query w-mer matches that are deemed redundant and non-redundant on the basis of a user-specified redundancy threshold N as well as the preferred design of a redundancy filter stage 114 is described in documents that were previously incorporated by reference. The output 130 from redundancy filter stage 114 is preferably a stream of ordered pairs as with stream 128, with the exception that the ordered pairs for matches that are deemed redundant with other ordered pairs are not present therein.

[0077] Also, it is worth noting that the implementation on an FPGA of the FPGA pipeline stages for the programmable logic device 102 described herein is within the skill of a person having ordinary skill in the art following the teachings herein and the teachings of (1) U.S. Patent No. 6,711,558 entitled "Associative Database Scanning and Information Retrieval"; (2) pending U.S. patent application 10/153,151, filed May 21, 2002 and entitled "Associative Database Scanning and Information Retrieval Using FPGA Devices" (published as U.S. patent application publication 2003/0018630); and (3) pending PCT application PCT/US04/16398 filed May 21, 2004 and entitled "Intelligent Storage and Processing Using FPGA Devices"; the entire disclosures of all of which have been incorporated herein by reference. For example, see the description for Figure 8 in PCT application PCT/US04/16398.

[0078] After the first stage (Stage 1) for word matching 54, being accelerated, as previously described and shown in Figure 1, the performance of the second stage (Stage 2) that relates to the ungapped extension 56 directly determines the performance of the overall pipelined application.

- [0079] The purpose of extending a w-mer is to determine, as quickly and accurately as possible, whether the w-mer arose by chance alone, or whether it may indicate a significant match. Ungapped extension in the second stage (Stage 2) 56 must decide whether each w-mer emitted from word matching is worth inspecting by the more computationally intensive, gapped extension. It is important to identify and discard spurious w-mers as early as possible in the BLAST pipeline because later stages are increasingly more complex. There exists a delicate balance between the stringency of the filter and its sensitivity, i.e. the number of biologically significant alignments that are found. A highly stringent filter is needed to minimize time spent in fruitless gapped extension, but the filter must not discard w-mers that legitimately identify long query-database matches with few differences. In the illustrative, but nonlimiting, embodiment of an implementation of a programmable logic device 102, e.g., FPGA, this filtering computation must also be parallelizable and simple enough to fit in a limited area.
- [0080] The implementation of the second stage (Stage 2) 56 builds on the concepts utilized in the implementation of the first stage (Stage 1) 54 described above and disclosed in P. Krishnamurthy, J. Buhler, R. D. Chamberlain, M. A. Franklin, K. Gyang, and J. Lancaster, *Biosequence Similarity Search On The Mercury System*, In Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP04), Pages 365-375 (2004), which is incorporated herein by reference in its entirety.
- [0081] The second stage (Stage 2) 56 deploys an ungapped extension stage utilizing a programmable logic device, e.g., FPGA 74, which operates as a prefilter stage. This design utilizes the speed of the implementation of the programmable logic device, e.g., FPGA 74, to greatly reduce the number of w-mers passed to software while retaining the flexibility of the software implementation on those w-mers that pass in the pipeline 70.
- [0082] Preferably, but not necessarily, a w-mer must pass the second stage (Stage 2) 56 ungapped extension, which may utilize both hardware and software, before being released to the third stage (Stage 3) gapped extension 58. By focusing on data streaming nature of the application with overall throughput and not latency, the addition of another processing stage is very useful.
- [0083] Utilizing a first illustrative software program, e.g., NCBI BLASTN, the second stage (Stage 2) 56 is now described. The ungapped extension of a w-mer into an HSP runs in two steps utilizing NCBI BLASTN. The w-mer is extended back toward the beginnings of the two sequences, then forward toward their ends. As the HSP extends over each character pair, that pair receives a reward $+\alpha$ if the characters match or a penalty $-\beta$ if they mismatch. An

HSP's score is the sum of these rewards and penalties over all its pairs. The end of the HSP in each direction is chosen to maximize the total score of that direction's extension. If the final HSP scores above a user-defined threshold, it is passed on to the third stage (Stage 3), gapped extension 58.

[0084] For long biological sequences, it is useful to terminate ungapped extension before reaching the ends of the sequences, especially if no high scoring HSP is likely to be found. In the illustrative software, BLASTN, implements early termination by an X-drop mechanism. The algorithm tracks the highest score achieved by any extension of the w-mer thus far; if the current extension scores at least X below this maximum, further extension in that direction is terminated. Ungapped extension with X-dropping allows the BLASTN software to recover HSPs of arbitrary length while limiting the average search space for a given w-mer. However, because the regions of extension can in principle be quite long, this heuristic is not as suitable for fast implementation in a typical programmable logic device 74, e.g., FPGA. Note that even though extension in both directions can be done in parallel, this was not sufficient to achieve a desired speedup on processing.

[0085] The prior processing of the second stage (Stage 2) 56 utilizing ungapped extension is shown in Figure 9 utilizing NCBI BLAST. The parameters used here are $w = 5$, $\alpha = 1$, $\beta = -3$, and $X\text{-drop} = 10$. W is the length of a substring, α is a positive reward for character match, β is a negative penalty for character mismatch and X-drop is the reduction below the maximum level of current extension scores where further extension is terminated. The use of X-dropping allows the software to recover HSPs of an arbitrary length while limiting the average search space for a given w-mer but it is still not very fast or efficient.

[0086] The ungapped extension analysis with a preferred software program, i.e., NCBI BLAST, is generally indicated by numeral 62 and begins at the end of the w-mer and extends left. The extension stops when the running score drops 10 below the maximum score (as indicated by the arrows). This is also indicated as the user modifiable "X-drop" parameter. The same computation is then performed in the other direction. The final substring is the concatenation of the best substrings from the left and right extensions.

[0087] In the present invention, a prefilter is utilized to improve processing efficiency. The extension algorithm for a given w-mer is performed in a single forward pass over a fixed size window. This extension begins by calculating the limits of a fixed window of length L_w , which is centered on the w-mer in both the query and database stream. Afterward, the appropriate substrings of the query and the database stream are then fetched into buffers. Once the substrings are buffered, then the extension algorithm begins. The extension

algorithm in pseudo code is shown on Figure 12 and generally indicated by numeral 66 and the associated flow chart is shown on Figure 20 and is generally indicated by numeral 500. This algorithm 66 also lends itself to hardware implementation despite the sequential expression of the computation.

[0088] This prefilter extension algorithm implements a dynamic programming recurrence that simultaneously computes the start and end of the best high-scoring segment pair (HSP) in a predetermined window. The first step 502 is to calculate predetermined window boundaries. The current position within a window is indicated by a variable " i ". There is then a reset of the score " γ " of the best high-scoring segment pair (HSP) that terminates at position i of the window and the score " T " of the best high-scoring segment pair (HSP) ending at or before i to zero (0). This is followed by a reset of the variable " B " as well as the two endpoints B_{\max} and E_{\max} of the best high scoring segment pair (HSP) ending at or before i to the value of zero (0).

[0089] First, for every position i in a window up to the length of the window L_w , a determination is made as to whether characters in the window match 504 and the score contribution of each character pair in the window is computed, using a reward $+\alpha$, indicated by numeral 506 and a penalty $-\beta$, indicated by numeral 508 as the implementation for providing rewards for pattern matches and penalties for pattern mismatches, respectively. These contributions can be calculated independently in parallel for each pair. If there is a reward $+\alpha$, it is added to the score γ , i.e., $\gamma = \gamma + \alpha$ and if there is a penalty $-\beta$, it is subtracted from the score, i.e., $\gamma = \gamma - \beta$.

[0090] Then, the algorithm determines if the score γ of the best high-scoring segment pair (HSP) is greater than zero (0) 510. If this determination is positive, then there is a determination if the score " γ " of the best high-scoring segment pair (HSP) that terminates at position i of the window is greater than the score " T " of the best high-scoring segment pair (HSP) ending at or before i and if i (position in the window) is greater than the end of a short word match w -mer (W_{merEnd}) 511.

[0091] If this determination in step 510 is negative, then there is a determination of whether i is less than the start of the short word match w -mer (W_{merStart}) 512. If i is less than the start of the short word match w -mer, then the variable B is set to the current position within a window i incremented by one (1) and the score " γ " of the best high-scoring segment pair (HSP) is set to (0) zero 516. The next step is then a determination if i is equal to the length of the window L_w 519. If the determination in step 511 is positive, then the score " T " of the best high-scoring segment pair (HSP) ending at or before i is set to the score " γ " of the best

high-scoring segment pair (HSP) that terminates at i , the endpoint B_{\max} of the best high-scoring segment pair (HSP) "T" ending at or before i is set to the variable B and the endpoint E_{\max} of the best high-scoring segment pair (HSP) is set to i (position in the window) 514. The next step is then a determination if i is equal to the length of the window L_w 519. Moreover, if the determination in step 511 is negative, then there is also a determination if i is equal to the length of the window L_w 519.

[0092] If the determination in step 519 is negative, then the variable i is incremented by one (1) in step 517 and the process is returned to step 504. If this determination in step 519 is positive, then the software program proceeds to the next program step 518, which is a determination whether the score of the best high-scoring segment pair (HSP) "T" is greater than the variable T in step 518. If this determination is positive, then a value of "true" is returned 520 and the software program ends 527.

[0093] If the determination in step 518 is negative, this is followed by a determination of whether the position of the endpoint B_{\max} of the best high-scoring segment pair (HSP) is equal to zero 522. If this determination is positive, then a value of "true" is returned 526 and the software program ends 527.

[0094] Finally, if the determination in step 522 is negative, there is a determination of whether the endpoint E_{\max} of the best high-scoring segment pair (HSP) is equal to the length of the window L_w 524. If this determination is positive, then a value of "true" is returned 526 and the algorithm ends 527 and if this determination is negative, then a value of "false" is returned 528 and the algorithm will again end 527.

[0095] In summary, the algorithm tracks T , which is the score of the highest scoring HSP ending before i , along with its endpoints B_{\max} and E_{\max} . Therefore, if FL_w is greater than a user-defined threshold score, the w -mer passes the prefilter and is forwarded to (Stage 2b) 84 for the ungapped extension.

[0096] There are two aspects of this prefilter algorithm. The first aspect is that the recurrence requires that the HSP found by the algorithm pass through its original matching w -mer and that a higher-scoring HSP in the window that does not contain this w -mer is ignored. This constraint ensures that, if two distinct biological features appear in a single window, the w -mers generated from each have a chance to generate two independent HSPs. Otherwise, both w -mers might identify only the feature with the higher-scoring HSP, causing the other feature to be ignored.

[0097] The second aspect is if the highest scoring HSP intersects the bounds of the window, it is passed on to software regardless of the score. This heuristic ensures that HSPs that might

extend well beyond the window boundaries are properly found by software, which has no fixed size window limits, rather than being prematurely eliminated.

[0098] An illustrative example of the ungapped extension prefilter utilizing the illustrative Mercury BLAST is shown in Figure 10. Using the same parameters of Figure 9 with the addition of a new parameter L_w , which is the length of a predetermined window, then $L_w = 19$, $w = 5$, $\alpha = 1$, $\beta = -3$, the present invention, e.g., Mercury BLAST, ungapped extension, as generally indicated by numeral 65, begins at the leftmost base of the window (indicated by brackets) and moves right, calculating the best-scoring substring in the window. In this example, the algorithms gave the same result in both Figures 9 and 10. However, this is not necessarily the situation in general.

[0099] An illustrative system of the present invention is shown in schematic form as the application pipeline and is generally indicated by numeral 70 is shown in Figure 11. This system is preferably an infrastructure designed to accelerate disk-based computations and exploits the high input/output (I/O) bandwidth available from modern disks by streaming data 52 directly from the disk medium to programmable logic devices such as, but not limited to, FPGAs (field programmable gate arrays) 74.

[00100] The streaming data is delivered to the hardware word matching prefilter 76, which is designated as Stage 1a. After the data stream is prefiltered, it passes into a word matching module 80, as previously described above and is designated as Stage 1b. The word matching module 80 utilizes memory 78, e.g., SRAM (static random access memory that retains data bits in its memory as long as power is being supplied and does not have to be periodically refreshed). The combination of Stage 1a and 1b, 76 and 80 form the previously described first stage (Stage 1) 54, that was previously shown in Figure 1.

[00101] The data then passes into an ungapped extension prefilter 82 which includes dedicated hardware and/or firmware, which is designated as Stage 2a. The output of the prefilter then passes into the remainder of Stage 2, which is the ungapped extension 84, which is designated as Stage 2b and includes software. The combination of Stage 2a and 2b, 82 and 84 form the previously described second stage (Stage 2) 56, previously shown in Figure 1.

[00102] In the alternative, if there are enough hardware/firmware resources available, the software ungapped extension 84, which is designated as Stage 2b, can be eliminated with the hardware/firmware ungapped extension prefilter 82 functioning as the sole ungapped extension, which is designated as Stage 2 and shown in Figure 11A. This methodology can provide higher throughput since the software portion of the ungapped extension can be

eliminated in the utilization of the biological sequence similarity search computation, e.g., BLAST, search.

[00103] The data stream finally passes into the gapped extension 58, which is designated as Stage 3. The ungapped extension and gapped extension can be operated by a wide variety of computing mechanisms, including, but not limited to, a processor 88. The prefilter (Stage 2a) 82 lends itself to hardware implementation despite the sequential expression of the computation in Figure 12.

[00104] The ungapped extension prefilter design 82 is fully pipelined internally and accepts one match per clock. The prefilter is parameterizable with the parameters chosen based on a number of design-specific constraints. The commands are supported to configure parameters such as a match score, a mismatch score, and a cutoff threshold. Therefore, there is a trade-off between sensitivity and throughput that is left to the discretion of the user.

[00105] Since the w-mer matching stage generates more output than input, two independent data paths are used for input into the ungapped extension, i.e., Stage 2a, which is generally indicated by numeral 82 and is shown in Figure 13. The w-mers/commands and the data are parsed with the w-mers/commands are received on one path 92, and the data from the database is received on the other path 93. The commands are supported by an active control valid signal and the commands are also transmitted in a predetermined format. Also, the commands are designed so that the programmable logic device 74, e.g., FPGA, shown in Figure 11, does not need to be reconfigured.

[00106] The module is organized into three (3) pipelined stages. This includes an extension controller 94, a window lookup module 95 and a scoring module 98. There is an extension controller 94 that parses the input to demultiplex the shared w-mers command 92 and database stream 93. A w-mer matches and the database stream flow through the extension controller 94 into the window lookup module 95. The window lookup module 95 is the module that is responsible for fetching the appropriate substrings of the database stream and the query to form an alignment window.

[00107] The window lookup module 95 is illustrated in additional structural detail in Figure 13A. Preferably, but not necessarily, a query is stored on-chip utilizing dual-ported random access memory (RAM) on the programmable logic device 74, e.g., FPGA, shown in Figure 11. The query is streamed 202 at the beginning of each biological sequence similarity search computation, e.g., BLAST search, and is fixed until the end of the database is reached. The size of the query stream 202 is limited to the amount of memory, e.g., block random access memories (RAMs) that are provided in the query buffer 204. The database is also streamed

206 in a similar manner as the query stream 202 and provided to a database buffer 208. The database buffer 208 is preferably, but not necessarily, a circular buffer. The use of a circular buffer is helpful to retain a necessary section of the database stream 206 that any windows that are formed from the arriving w-mers might require. Since the w-mer generation is performed in the first stage 54, as shown in Figure 1, only a small portion of the database stream 206 needs to be buffered in the database buffer 208 in order to accommodate each extension request.

[00108] The database buffer 208 is preferably, but not necessarily, built to allow a fixed distance of w-mers that are out-of-order to be processed correctly. For the previously referenced BLASTP implementation, this is important since the w-mers in the word matching stage 80, as shown in Figure 11, will possibly be out-of-order.

[00109] The window lookup module 95 preferably, but not necessarily, is organized as a six (6) stage pipeline that is shown in Figure 13A. This includes a first stage 210, a second stage 212, a third stage 214, a fourth stage 216, a fifth stage 218 and a sixth stage 220. Extensive pipelining may be necessary to keep up with requests from the previous stage, i.e., extension controller 94 shown in Figure 13, which may come once per clock pulse.

[00110] The first stage of the pipeline 210 calculates the beginning of the query and database windows based on the incoming w-mer and a configurable window size. The offset of the query 222 and the offset of the database 224 is provided to the first stage 210. This first stage of the pipeline 210 calculates the beginning of the query window and database windows based on the incoming w-mer and a configurable window size. The offset of the beginning of the window is then passed to the buffer modules, i.e., the second stage 212, the third stage 214, the fourth stage 216 as well as the fifth stage 218, which begins the task of reading a superset of the window from memory, e.g., block RAMs. An extra word of data must be retrieved from the memory, e.g., block RAMs, because there is no guarantee that the window boundaries will fall on a word boundary. Therefore, an extra word is fetched on each lookup so that the exact window can be constructed from a temporary buffer holding a window size worth of data plus the extra word. The next four (4) stages, i.e., the second stage 212, the third stage 214, the fourth stage 216 and the fifth stage 218, move the input data lock-step with the buffer lookup process and ensure that pipeline stalls are handled in a correct manner. Finally, the superset of the query and database windows arrive to the final stage or sixth stage for format output 220. The output includes a seed out 226, a query window 228 and a database window 230 so that the correct window of the buffers is extracted and registered as the output.

[00111] In an illustrative, but nonlimiting, embodiment, the query is buffered on-chip using the dual-ported block random access memory (BRAM) on to programmable logic devices such as, but not limited to, FPGAs (field programmable gate arrays) 74, as shown in Figure 11. Preferably, but not necessarily, the database stream is preferably buffered in a circular buffer, which can be created from BRAMs. As previously mentioned, the w-mer generation is done in the first stage for the extension controller 94. Only a small amount of the database stream 93 needs to be buffered to accommodate each input w-mer because they arrive from Stage Ia, 76, and Stage Ib, 80, in order with respect to the database stream 93.

[00112] Since the illustrative, but nonlimiting, dual-ported block random access memories (BRAMs) are a highly-utilized resource in Stage Ia, 76, and Stage Ib, 80, the memories, e.g., dual-ported block random access memories BRAMs, are time-multiplexed to create a quad ported BRAM structure. After the window is fetched, it is passed into the scoring module 98 and stored in registers. The scoring module 98 implements the recurrence of the extension algorithm 66 on Figure 12. Since the computation is too complex to be done in a single cycle, the scorer is extensively pipelined.

[00113] The first stage of the scoring pipeline 98 is shown in Figure 13. The base comparator 96 receives every base pair in parallel registers as shown by numerals 302 and 304, respectively as shown in Figures 13 and 14. These pairs are fed into a plurality of comparators 306 that assign a comparison score to each base pair in the window. In the illustrative but preferred embodiment, for DNA sequences, e.g., BLASTN, the base comparator 96 assigns a reward $+\alpha$, generally indicated by numeral 310, to each matching base pair and a penalty $-\beta$, generally indicated by numeral 312 to each mismatching pair. The score computation is the same for protein sequence analysis, e.g., BLASTP, except there are many more choices for the score. In BLASTP, the reward $+\alpha$ and the penalty $-\beta$ are replaced with a value from a lookup table that is indexed by the concatenation of the two symbols 302, 304. The calculation of all of the comparison scores is performed in a single cycle using L_w comparators. After the scores are calculated, the calculated scores are delivered to later scoring stages 309.

[00114] The scoring module 98 is preferably, but not necessarily, arranged as a classic systolic array. The data from the previous stage are read on each clock pulse and results are output to the following stage on the next clock pulse. Storage for comparison scores in successive pipeline stages 97 decrease in every successive stage and is shown in Figure 13. This decrease is possible because the comparison score for window position "z" is consumed in the

ith pipeline stage and may then be discarded, since later stages inspect only window positions that are greater than i .

[00115] This structure of data movement is shown in more detail in Figure 15, which is generally indicated by numeral 97 as a systolic array of scoring stages. The darkened registers 326 hold the necessary comparison scores for the w-mer being processed in each pipeline stage. Although Figure 15 shows a single comparison score being dropped for each scorer stage for ease of explanation, the preferred embodiment includes dropping two comparison scores per stage. There is also a plurality of pipeline scoring stages 330. The data flows from left to right on Figure 15. The light registers 328 are pipeline calculation registers and are used to transfer the state of the recurrence from a previous scoring stage to the next with each column of registers containing a different w-mer in the pipeline 70. Initialization 322 is provided to the light registers 328, dark registers 326 and scoring stages 330.

[00116] The interface of an individual scoring stage is generally indicated by numeral 330 and is shown in Figure 16. The values shown entering the top of the scoring stage 330 are the state of dynamic programming recurrence propagated from the previous scoring stage. These values are read as input to a first register 342 and the results are stored in the output registers 364 shown on the right. Each scoring stage 362 in the pipeline 97 contains combinational logic to implement the dynamic programming recurrence shown in Lines 12-19 of the algorithm described in Figure 12. This includes a value for γ indicated by numeral 344, a value for B indicated by numeral 346, a value for T indicated by numeral 348, a maximum value of B , i.e., B_{\max} , indicated by numeral 350 and a maximum value of E_5 i.e., E_{\max} , indicated by numeral 352.

[00117] The data entering from the left of the second register 360 are the comparison scores 358, i.e., α/β , and the database offset 354 and query offset 356 for a given w-mer, which are independent of the state of the recurrence. In order to sustain a high clock frequency design, each scoring stage computes only two iterations of the loop per clock cycle, resulting in $L_w/2$ scoring stages for a complete calculation. This provides step 1, indicated by numeral 361 and step 2, indicated by numeral 363, which implements two iterations of Lines 12 through 19 of the algorithm 66 shown in Figure 12. This is merely an illustrative, but nonlimiting, number of loop iterations within a single stage since any number of loop iterations can be utilized. This tradeoff is preferred when contrasting area and speed utilizing the preferred and illustrative, but nonlimiting, hardware. Therefore, there are $L_w/2$ independent w-mers being

processed simultaneously in the scoring stages 362 of the processor when the pipeline 70 is full.

[00118] The pipeline calculation registers output 364 that are provided to subsequent scoring stages, e.g., $i + 1$. This includes a value for γ indicated by numeral 366, a value for B indicated by numeral 368, a value for T indicated by numeral 370, a maximum value of B , i.e., B_{\max} , indicated by numeral 372 and a maximum value of E , i.e., E_{\max} , indicated by numeral 374.

[00119] The final pipeline stage of the scoring module is the threshold comparator 99 that is shown in Figure 13. The comparator takes the fully-scored segment and makes a decision to discard or keep the segment. This decision is based on the score of the alignment relative to a user-defined threshold T , as well as the position of the highest-scoring substring. If the maximum score is above the threshold, the segment is passed on. Additionally, if the maximal scoring substring intersects either boundary of the window, the segment is also passed on, regardless of the score. If neither condition holds, the substring of a predetermined length, i.e., segment, is discarded. The segments that are passed on are indicated by numeral 100 on Figure 13.

[00120] In an optional and illustrative implementation, the ungapped extension prefilter 82, as shown in Figure 13, utilizes approximately 38% of the logic cells and 27 BRAMs on a reconfigurable hardware such as, but not limited to, FPGAs (field programmable gate arrays) 74. An illustrative, but nonlimiting, example includes a Xilinx® Virtex-II 6000® series FPGA. Xilinx, Inc., is a Delaware Corporation, having a place of business at 2100 Logic Drive, San Jose, California 95124-3400. This includes the infrastructure for moving data in and out of the reconfigurable hardware, e.g., FPGA, itself. The illustrative, but nonlimiting, current design runs at 96 MHz, processing one w-mer per clock. The full Mercury BLASTN design utilizes approximately 65% of the logic cells and 134 BRAMs.

[00121] There are many aspects to the performance of an ungapped extension prefilter 82, as shown in Figure 13. First is the individual stage throughput. The ungapped extension stage (Stage 2a) 82 must run fast enough to not be a bottleneck in the overall pipeline 70. Second, the ungapped extension stage (Stage 2a) 82 must effectively filter as many w-mers as possible, since downstream stages are even more computationally expensive. Finally, the above must be achieved without inadvertently dropping a large percentage of the significant alignments (i.e., the false negative rate must be limited).

[00122] First, throughput performance for the ungapped extension stage alone is a function of data input rate. In an illustrative, but nonlimiting, example, the ungapped extension stage

accepts one w-mer per clock and runs at 96 MHz. Hence the maximum throughput of the filter is one (1) input match/cycle times 96 MHz = 96 Mmatches/second. This provides a speedup of twenty-five (25) over the software ungapped extension 82 executed on the previously described baseline system, as shown in Figure 1. This analysis is based solely on modeling, based on assumed parameters and constraints, and actual results can vary.

[00123] Figure 17 shows the application throughput (quantified by the ingest rate of the complete pipeline 70, in million bases per second) as a function of the performance attainable in the second stage (Stage 2) 56, as shown in Figure 1, which is quantified by the ingest rate of the second stage (Stage 2) 56 alone, in million matches per second). Figure 17 includes results for both 25,000-base double stranded queries and 20,000-base double stranded queries.

[00124] Figure 18 plots the resulting speedup versus throughput (for both query sizes) for the preferred embodiment of the present invention with a prefiltering stage (Stage 2a) 82 as shown on Figure 11 relative to the software implementation. Once again, this analysis is based solely on modeling, based on assumed parameters and constraints, and actual results can vary. The software profiling shows, for 25,000-base queries, an average execution time for the second stage (Stage 2) 56 alone of 0.265 μ s/match. This corresponds to a throughput of 3.8 Mmatches/s, plotted towards the left in Figures 17 and 18. As the performance of the second stage (Stage 2) 56 is increased, the overall pipeline 70 performance increases proportionately until the second stage (Stage 2) 56 is no longer the bottleneck stage.

[00125] To explore the impact this performance in the second stage (Stage 2) 56 has on the overall system, we return to the graphs of Figures 17 and 18. Above approximately 35 Mmatches/s, the overall system throughput is at its maximum rate of 1400 Mbases/s, with a speedup of forty-eight (48 times over that of software, e.g., NCBI BLASTN, for a 25,000-base query) and a speedup of thirty-eight (38 times over that of software, e.g., NCBI BLASTN, for a 20,000-base query).

[00126] There are two (2) parameters of ungapped extension that affect its sensitivity. The first parameter is the score threshold used, which affects the number of HSPs that are produced. The second parameter is the length of the window, which can affect the number of false negatives that are produced. The HSPs that have enough mismatches before the window boundary to be below the score threshold but have many matches immediately outside the boundary will be incorrectly rejected.

[00127] To evaluate the functional sensitivity of hardware ungapped extension 82, measurements were performed using an instrumented version of software, e.g., NCBI

BLASTN. A software emulator of the new ungapped extension algorithm was placed in front of the standard ungapped extension stage, e.g., NCBI. Then, the statistics were gathered which show how many w-mers arrived at the ungapped extension prefilter stage 82, and how many passed.

[00128] These statistics were collected both for ungapped extension, the second stage (Stage 2) 56, Figure 1, alone, e.g., NCBI BLASTN, and with the hardware emulator in place. The dataset was generated from the human and mouse genomes. The queries were statistically significant samples of various sizes (e.g., 10 kbase, 100 kbase, and 1 Mbase). The database stream was the mouse genome with low-complexity and repetitive sequences removed.

[00129] Figure 19 is a table, generally indicated by numeral 380, which summarizes the results for a window size of 64 bases, which is the window size used in the illustrative, but nonlimiting hardware implementation. A score threshold of twenty (20) corresponds to the default value in ungapped extension or second stage (Stage 2) 82, Figure 11, e.g., NCBI BLASTN. The reject fraction is the measured ratio of output HSPs over input w-mers. This value quantifies the effectiveness of the overall second stage (Stage 2) 82, Figure 11, at filtering w-mers so they need not be processed in (Stage 2b) 84. The percent found is the percentage of gapped alignments present in the MERCURY BLAST'S output relative to NCBI BLASTN output, as shown in Figure 19.

[00130] Using a window length of 64 bases, the ungapped extension prefilter 82 is able to filter out between 99.5% and 99.99% of all its input. For instance, a threshold of seventeen (17) missed HSPs provides a good tradeoff between a high reject fraction while keeping the vast majority (99.95%) of the significant HSPs. This translates into five (5) missed HSPs out of 10,334 HSPs found by software system, e.g., NCBI BLAST.

[00131] Therefore, biosequence similarity searching can be accelerated practically by a pipeline 70 designed to filter high-speed streams of character data utilizing a performance-critical ungapped extension stage 56. The present invention is a highly parallel and pipelined implementation yields results comparable to those obtained from software, such as BLASTN, while running twenty-five times faster and enabling the entire accelerator to run approximately forty to fifty times faster.

[00132] For example, the techniques of the present invention can be used to implement word matching for seeded alignment searching techniques other than biological sequence similarity searching. Furthermore, the techniques of the present invention can also be applied to BLAST versions other than BLASTN, such as BLASTP, BLASTX, TBLASTN, and TBLASTX. Further still, the preferred embodiment has been described wherein the starting

position of each matching w-mer within either the database sequence or the query sequence is determined by the various pipeline stages. However, it should be noted that these positions need not necessarily be the starting position. For example, the ending position or some intermediate position can also be used to identify the positions of the matching w-mers within the database and query sequences. These changes and modifications should be considered as alternative embodiments for the present invention, and the invention should be considered as limited only by the scope of the claims appended hereto and their legal equivalents.

[00133] Thus, there has been shown and described several embodiments of a novel invention. As is evident from the foregoing description, certain aspects of the present invention are not limited by the particular details of the examples illustrated herein, and it is therefore contemplated that other modifications and applications, or equivalents thereof, will occur to those skilled in the art. The terms "have," "having," "includes" and "including" and similar terms as used in the foregoing specification are used in the sense of "optional" or "may include" and not as "required." Many changes, modifications, variations and other uses and applications of the present construction will, however, become apparent to those skilled in the art after considering the specification and the accompanying drawings. All such changes, modifications, variations and other uses and applications which do not depart from the spirit and scope of the invention are deemed to be covered by the invention which is limited only by the claims that follow.

CLAIMS

1. A digital logic circuit for performing biological sequence similarity searching, the circuit comprising:

a programmable logic device configured to include a pipeline comprising a Bloom filter stage, the Bloom filter stage being configured to receive a data stream comprising a plurality of biological sequence data strings and filter the biological sequence data stream to identify a plurality of possible matches between the sequence data strings and a plurality of substrings of a query string.

2. The circuit of claim 1, wherein the programmable logic device comprises a FPGA.

3. The circuit of claim 2, wherein the biological sequence similarity searching pipeline comprises a first stage for BLAST.

4. The circuit of claim 3, wherein the BLAST first stage comprises a first stage for BLASTN.

5. The circuit of claim 3, wherein the BLAST first stage comprises a first stage for one selected from the group consisting of BLASTP, BLASTX, TBLASTN₅ and TBLASTX.

6. The circuit of claim 3, wherein the Bloom filter stage comprises a plurality of parallel Bloom filters.

7. The circuit of claim 6, wherein the plurality of parallel Bloom filters are programmed with a plurality of query substrings prior to processing the biological sequence data stream through the pipeline.

8. The circuit of claim 7, wherein each of the plurality of parallel Bloom filters are programmed with the same query substrings.

9. The circuit of claim 8, wherein each Bloom filter is configured to access a dual port BRAM on the FPGA, wherein the dual port BRAM is double clocked to replicate a four port BRAM.

10. The circuit of claim 9, wherein access to each dual port BRAM is shared by four parallel Bloom filters.

11. The circuit of claim 8, wherein each Bloom filter is configured to access a dual port BRAM on the FPGA, wherein the dual port BRAM is triple clocked to replicate a six port BRAM.

12. The circuit of claim 11, wherein access to each dual port BRAM is shared by at least six parallel Bloom filters.

13. The circuit of claim 8, wherein each Bloom filter is configured to utilize k hash functions and $k(1 \times m)$ bit memory units, each memory unit corresponding to one of that Bloom filter's hash functions, each hash function being configured to map received sequence strings to bits in its corresponding memory unit to determine if a possible match exists.

14. The circuit of claim 13, wherein each memory unit comprises a dual port BRAM unit.

15. The circuit of claim 13, wherein access to each dual port BRAM unit is shared by a plurality of the Bloom filters.

16. The circuit of claim 15, wherein each dual port BRAM unit is double clocked such that it is shared by at least four of the Bloom filters.

17. The circuit of claim 7, wherein the pipeline further comprises a hashing stage downstream from the Bloom filter stage, the hashing stage being configured to map the possible matches to at least one position within the query string.

18. The circuit of claim 17, further comprising a memory in communication with the hashing stage, the memory being configured to store a hash table, the hash table storing at least one pointer to a position in the query string for a possible match.

19. The circuit of claim 17, wherein the hashing stage comprises a perfect hashing stage.

20. The circuit of claim 17, wherein the hashing stage comprises a near perfect hashing stage.

21. The circuit of claim 20, wherein the near perfect hashing stage is configured with functions A and B selected from the H3 family of hash functions, wherein A and B are guaranteed to be of full rank.

22. The circuit of claim 17, further comprising a redundancy filter downstream from the hashing stage, the redundancy filter being configured to receive data representative of possible matches from the hashing stage and remove possible matches that are redundant with previous possible matches within a selectable degree of redundancy.

23. The circuit of claim 22, wherein the redundancy filter is implemented on the FPGA.

24. The circuit of claim 3, further comprising a database in communication with the FPGA, the database being configured to store a biological sequence that is to be streamed through the FPGA.

25. A system for performing seeded alignment searching, the system comprising:
a programmable logic device configured to implement a seeded alignment searching pipeline, the pipeline comprising a Bloom filter stage, the Bloom filter stage being configured to receive a data stream, the data stream comprising a plurality of data strings and filter the data stream to identify a plurality of possible matches between the stream data strings and a plurality of substrings of a query string.

26. The system of claim 25, wherein the programmable logic device comprises a field programmable gate array (FPGA).

27. A method for performing biological sequence similarity searching, the method comprising:

processing a biological sequence data stream through a Bloom filter stage implemented on a programmable logic device, the biological sequence data stream comprising a plurality of biological sequence data strings, the Bloom filter stage comprising a Bloom filter that is programmed with a plurality of substrings of a query string and configured to identify a plurality of possible matches between the biological sequence data strings and the query substrings.

28. The method of claim 27, wherein the Bloom filter is further configured to identify a position of each possible match within the biological sequence data stream.

29. The method of claim 28, wherein the programmable logic device comprises a FPGA.

30. The method of claim 29, further comprising performing the processing step as a portion of a first stage for BLAST.

31. The method of claim 30, wherein the BLAST first stage comprises a first stage for BLASTN.

32. The method of claim 30, wherein the BLAST first stage comprises a first stage for one selected from the group consisting of BLASTP, BLASTX, TBLASTN, and TBLASTX.

33. The method of claim 30, wherein the Bloom filter stage comprises a plurality of parallel Bloom filters.

34. The method of claim 33, further comprising:
programming the parallel Bloom filters with a plurality of query substrings prior to processing the biological sequence data stream through the Bloom filter stage.

35. The method of claim 33, wherein the programming step comprises programming each of the parallel Bloom filters with the same query substrings.

36. The method of claim 35, wherein the Bloom filter stage comprises a plurality of dual port BRAM units, the method further comprising double clocking the BRAM units to replicate a four port BRAM units.

37. The method of claim 36, wherein each dual port BRAM unit is accessed by four parallel Bloom filters.

38. The method of claim 35, wherein the Bloom filter stage comprises a plurality of dual port BRAM units, the method further comprising triple clocking the BRAM units to replicate a six port BRAM units.

39. The method of claim 38, wherein each dual port BRAM unit is accessed by six parallel Bloom filters.

40. The method of claim 34, further comprising mapping, via a hashing stage, each possible match from the Bloom filter stage to a position in the query string corresponding to the query substring that caused the possible match.

41. The method of claim 40, wherein the mapping step is performed at least partially by the FPGA.

42. The method of claim 41, wherein the mapping step comprises performing the mapping via a perfect hashing stage.

43. The method of claim 41, wherein the mapping step comprises performing the mapping via a near perfect hashing stage.

44. The method of claim 43, wherein the near perfect hashing stage is configured with functions A and B selected from the H3 family of hash functions, wherein A and B are guaranteed to be of full rank.

45. The method of claim 40, further comprising removing possible matches that are redundant with other possible matches within a selectable degree of redundancy.

46. The method of claim 45, wherein the removing step is performed at least partially by the FPGA.

47. A system for generating a hash table for use in mapping a set of strings to keys, the system comprising:

a processor configured to provide near perfect hashing on the set of strings to identify a location in a hash table, utilizing an H3 family of hash functions, corresponding to each string and to generate the hash table, the hash table being configured to store, at each identified location therein is a key; and

a memory in communication with the processor for storing the hash table, wherein the processor is configured to provide near perfect hashing via hash functions, utilizing the H3 family of hash functions, that are guaranteed to be of full rank.

48. A system for generating a hash table for use in mapping a set of substrings to a position in a larger string, the system comprising:

a processor configured to provide near perfect hashing on the set of substrings to identify a position in a hash table, corresponding to each substring and to generate the hash table, the hash table being configured to store, at each identified location therein, a pointer to a position of each substring in the larger string; and

a memory in communication with the processor for storing the hash table.

49. A digital logic circuit for performing biological sequence similarity searching, the circuit comprising:

a programmable logic device configured to include a pipeline that comprises a matching stage, the matching stage being configured to receive a data stream comprising a plurality of possible matches between a plurality of biological sequence data strings and a plurality of substrings of a query string and the pipeline further comprises a prefilter stage located downstream from the matching stage, the ungapped extension prefilter stage being configured to shift through pattern matches between the biological sequence data strings and the plurality of substrings of a query string and provide a score so that only pattern matches

that exceed a user defined score will pass downstream from the ungapped extension prefilter stage.

50. The circuit of claim 49, wherein the programmable logic device comprises an FPGA.

51. The circuit of claim 49, wherein the matching stage comprises at least a portion of first stage for BLAST and the ungapped extension prefilter stage comprises at least a portion of a second stage for BLAST.

52. The circuit of claim 51, wherein the BLAST first stage is selected from the group consisting of a first stage for BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX and the BLAST second stage is selected from the group consisting of a second stage for BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX.

53. The circuit of claim 49, wherein the word matching stage further comprises a Bloom filter stage.

54. The circuit of claim 53, wherein the Bloom filter stage includes a plurality of Bloom filters.

55. The circuit of claim 53, wherein the word matching stage further comprises a hashing stage downstream from the Bloom filter stage, the hashing stage being configured to map the possible matches to at least one position within the query string.

56. The circuit of claim 55, further comprising a redundancy filter downstream from the hashing stage, the redundancy filter being configured to receive data representative of possible matches from the hashing stage and remove possible matches that are redundant with previous possible matches within a selectable degree of redundancy.

57. The circuit of claim 49, wherein the ungapped extension prefilter stage comprises an extension controller, a window lookup module and a scoring module.

58. The circuit of claim 49, wherein the ungapped extension prefilter stage utilizes parameters selected from the group consisting of a matching score, a mismatching score and a cutoff threshold.

59. The circuit of claim 49, wherein the ungapped extension prefilter stage comprises an extension controller.

60. The circuit of claim 59, wherein the extension controller comprises at least one of hardware, firmware and software.

61. The circuit of claim 59, wherein the extension controller comprises a plurality of data inputs including at least one first input for plurality of possible pattern matches of a predetermined length between a plurality of biological sequence data strings and a plurality of substrings of a query string and at least one second input for biological sequence data.

62. The circuit of claim 61, wherein at least one second input can receive commands for controlling parameters selected from the group consisting of a match score, a mismatch score or a cutoff threshold.

63. The circuit of claim 57, wherein the extension controller parses the biological data received by the at least one second input to demultiplex a shared command and the pattern matches of a predetermined length prior to submission into the window lookup module.

64. The circuit of claim 49, wherein the ungapped extension prefilter stage comprises a window lookup module.

65. The circuit of claim 64, wherein the window lookup module can fetch at least one substring of the biological sequence data and at least one substring of a query string to form a predefined window of biological data that can be utilized for analysis.

66. The circuit of claim 64, wherein the window lookup module receives at least one substring of the biological sequence data in a database buffer, at least one substring of a query string in a query buffer, an offset for at least one substring of the biological sequence

data and an offset for at least one substring of a query string, wherein the window lookup module calculates a beginning of a window that is passed to a plurality of buffer modules, wherein the plurality of buffer modules move the at least one substring of a query string and the at least one substring of the biological sequence data to an output stage to create at least one predefined window of biological data that can be utilized for analysis.

67. The circuit of claim 65, wherein the at least one substring of the biological sequence data and at least one substring of a query string are buffered on a plurality of BRAMS.

68. The circuit of claim 49, wherein the ungapped extension prefilter stage comprises a scoring module.

69. The circuit of claim 68, wherein the scoring module is capable of fetching biological data within a predefined window.

70. The circuit of claim 69, wherein the scoring module utilizes at least one iteration of algorithm instructions.

71. The circuit of claim 68, wherein the scoring module further comprises a base comparator, a plurality of scoring stages and a threshold comparator.

72. The circuit of claim 68, wherein the scoring module further comprises a base comparator.

73. The circuit of claim 72, wherein the base comparator receives at least one pair of a substring of the biological sequence data and a substring of a query string and compares these values and obtains at least one comparison score.

74. The circuit of claim 73, wherein the at least one comparison score includes a positive value for each matching pair and a negative score for each mismatching pair.

75. The circuit of claim 73, wherein the at least one comparison score is substituted from values located on a lookup table.

76. The circuit of claim 68, wherein the scoring module further comprises a plurality of scoring stages.

77. The circuit of claim 76, further comprising discarding at least one comparison score from each subsequent downstream scoring stage of the plurality of scoring stages.

78. The circuit of claim 76, wherein at least one scoring stage of the plurality of scoring stages receives input selected from the group of comparisons of biological sequence data in addition to biological sequence database and query positions within a predetermined window of data and at least one scoring stage of the plurality of scoring stages receives input from at least one other scoring stage of the plurality of scoring stages, wherein the input is selected from the group consisting of scores of high-scoring segment pairs and endpoints of high-scoring segment pairs.

79. The circuit of claim 68, wherein the scoring module further comprises a threshold comparator.

80. The circuit of claim 79, wherein the threshold comparator obtains a pattern match of a predetermined length that includes at least one score and determines if the at least one score exceeds a predetermined threshold value and passes the pattern match downstream.

81. The circuit of claim 79, wherein the threshold comparator obtains a pattern match of a predetermined length and determines if a high scoring substring intersects a boundary of a predetermined window of data.

82. A method for performing biological sequence similarity searching, the method comprising:

processing a biological sequence data stream with a programmable logic device configured to include a pipeline that comprises a matching stage, the matching stage being configured to receive a data stream comprising a plurality of possible matches between a plurality of biological sequence data strings and a plurality of substrings of a query string; and utilizing a ungapped extension prefilter stage located downstream from the matching stage in the pipeline, the ungapped extension prefilter stage being configured to

shift through pattern matches between the biological sequence data strings and the plurality of substrings of a query string and provide a score so that only pattern matches that exceed a user defined score will pass downstream from the ungapped extension prefilter stage.

83. The method of claim 82, wherein the programmable logic device comprises an FPGA.

84. The method of claim 82, wherein the matching stage includes utilizing at least a portion of a first stage for BLAST and the ungapped extension prefilter stage includes utilizing at least a portion of a second stage for BLAST.

85. The method of claim 84, wherein the BLAST first stage is selected from the group consisting of a first stage for BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX and the BLAST second stage is selected from the group consisting of a second stage for BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX.

86. The method of claim 82, comprises utilizing a Bloom filter stage in the matching stage of the pipeline.

87. The method of claim 86, wherein the Bloom filter stage includes a plurality of Bloom filters.

88. The method of claim 86, further comprising utilizing a hashing stage downstream from the Bloom filter stage and mapping the possible matches to at least one position within the query string with the hashing stage.

89. The method of claim 88, further comprising utilizing a redundancy filter downstream from the hashing stage, with the redundancy filter being configured to receive data representative of possible matches from the hashing stage and further comprising removing possible matches that are redundant with previous possible matches within a selectable degree of redundancy.

90. The method of claim 82, wherein the ungapped extension prefilter stage comprises an extension controller, a window lookup module and a scoring module.

91. The method of claim 82, further comprising utilizing parameters selected from the group consisting of a match score, a mismatch score or a cutoff threshold with the ungapped extension prefilter stage.

92. The method of claim 82, wherein the ungapped extension prefilter stage comprises an extension controller.

93. The method of claim 92, wherein the extension controller comprises at least one of hardware, firmware and software.

94. The method of claim 92, further comprising receiving a plurality of possible pattern matches of a predetermined length between a plurality of biological sequence data strings and a plurality of substrings of a query string as at least one first input and receiving biological sequence data with at least one second input.

95. The method of claim 94, further comprising receiving commands for controlling parameters selected from the group consisting of a match score, a mismatch score or a cutoff threshold with the at least one second input.

96. The method of claim 90, further comprising parsing the data received by the at least one second input to demultiplex the shared command and the pattern matches of a predetermined length prior to submission into the window lookup module.

97. The method of claim 82, wherein the ungapped extension prefilter stage comprises a window lookup module.

98. The method of claim 97, further comprising fetching at least one substring of the biological sequence data and at least one substring of a query string to form a predefined window of biological data for analysis with the window lookup module.

99. The method of claim 97, further comprising:

receiving at least one substring of the biological sequence data in a database buffer with the window lookup module;

receiving at least one substring of a query string in a query buffer with the window lookup module;

receiving an offset for the at least one substring of the biological sequence data and an offset for the at least one substring of a query string that is utilized to calculate a beginning of a window;

moving the offset for the at least one substring of the biological sequence data and the offset for at least one substring of a query string with a plurality of buffer modules; and

creating at least one predefined window of biological data for analysis based on the at least one substring of the biological sequence data, the at least one substring of a query string, the offset for the at least one substring of the biological sequence data and the offset for at least one substring of a query string.

100. The method of claim 98, further comprising buffering the at least one substring of the biological sequence data and at least one substring of a query string on a plurality of BRAMS.

101. The method of claim 82, wherein the ungapped extension prefilter stage comprises a scoring module.

102. The method of claim 101, further comprising fetching biological data within a predefined window with the scoring module.

103. The method of claim 101, further utilizing at least one iteration of algorithm instructions with the scoring module.

104. The method of claim 101, wherein the scoring module further comprises a base comparator, a plurality of scoring stages and a threshold comparator.

105. The method of claim 101, wherein the scoring module further comprises a base comparator.

106. The method of claim 105, further comprising receiving at least one pair of a substring of the biological sequence data and a substring of a query string and comparing these values and obtaining at least one comparison score with the base comparator.

107. The method of claim 106, wherein the at least one comparison score includes a positive value for each matching pair and a negative score for each mismatching pair.

108. The method of claim 106, further comprising substituting the at least one comparison score from values located on a lookup table.

109. The method of claim 101, wherein the scoring module further comprises a plurality of scoring stages.

110. The method of claim 109, further comprising discarding at least one comparison score for each stage of the plurality of scoring stages.

111. The method of claim 109, further comprising receiving input selected from the group of comparisons of biological sequence data and biological sequence database and query positions within a predetermined window of data with at least one scoring stage of the plurality of scoring stages and further comprising receiving input is selected from the group consisting of scores of high-scoring segment pairs and endpoints of high-scoring segment pairs from at least one other scoring stage of the plurality of scoring stages.

112. The method of claim 101, wherein the scoring module further comprises a threshold comparator.

113. The method of claim 112, further comprising obtaining a pattern match of a predetermined length that includes at least one score and determines if the at least one score exceeds a predetermined threshold value with the threshold comparator and passing the pattern match downstream.

114. The method of claim 112, further comprising obtaining a pattern match of a predetermined length and determines if a high scoring substring intersects a boundary of a predetermined window of data with the threshold comparator.

1/23

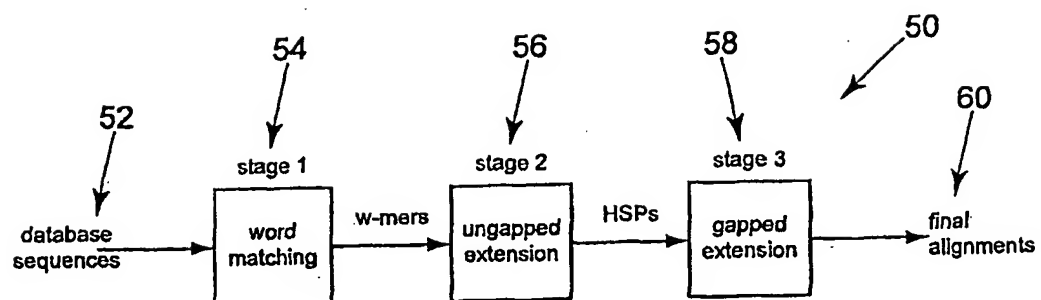


Figure 1

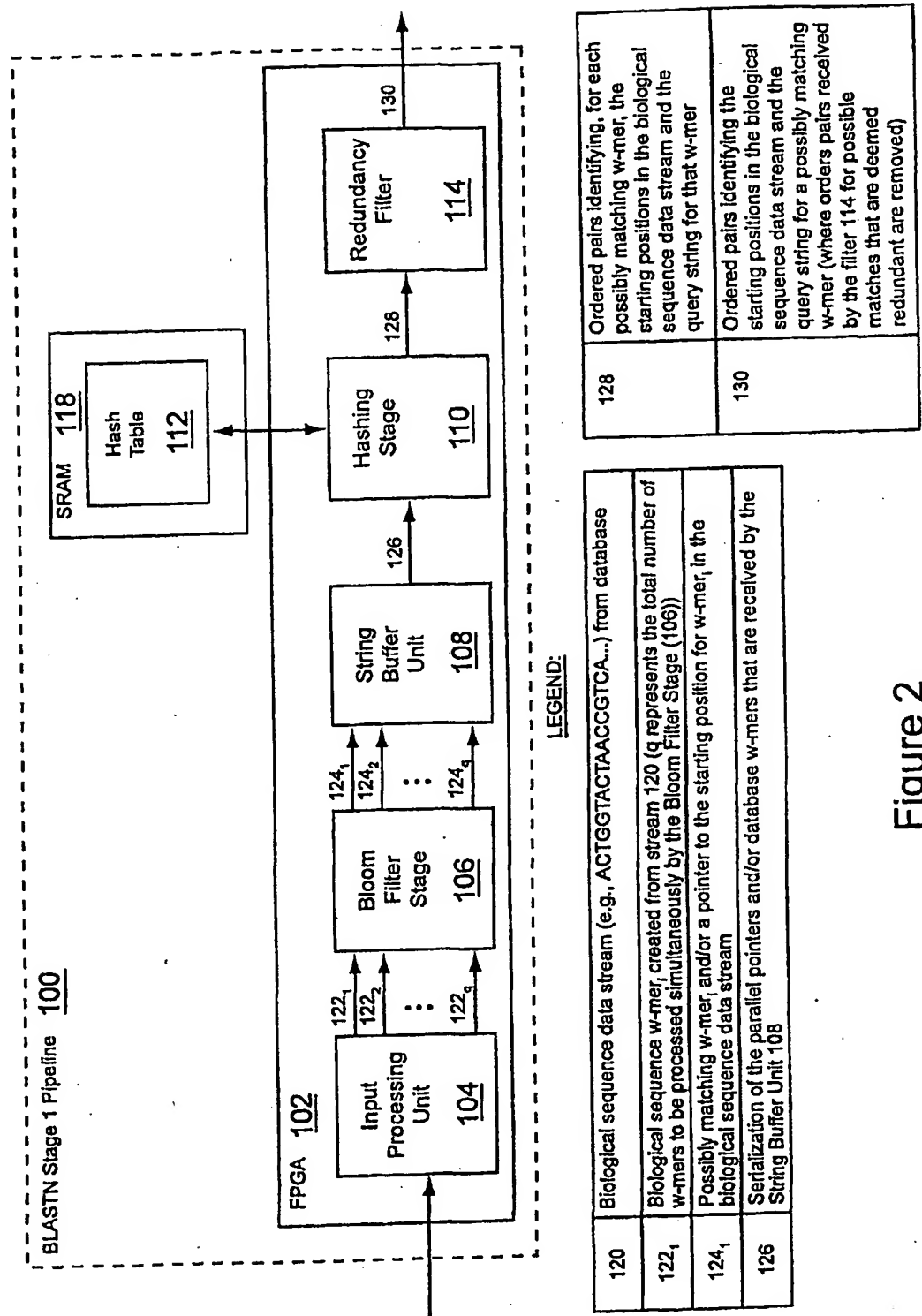


Figure 2

3/23

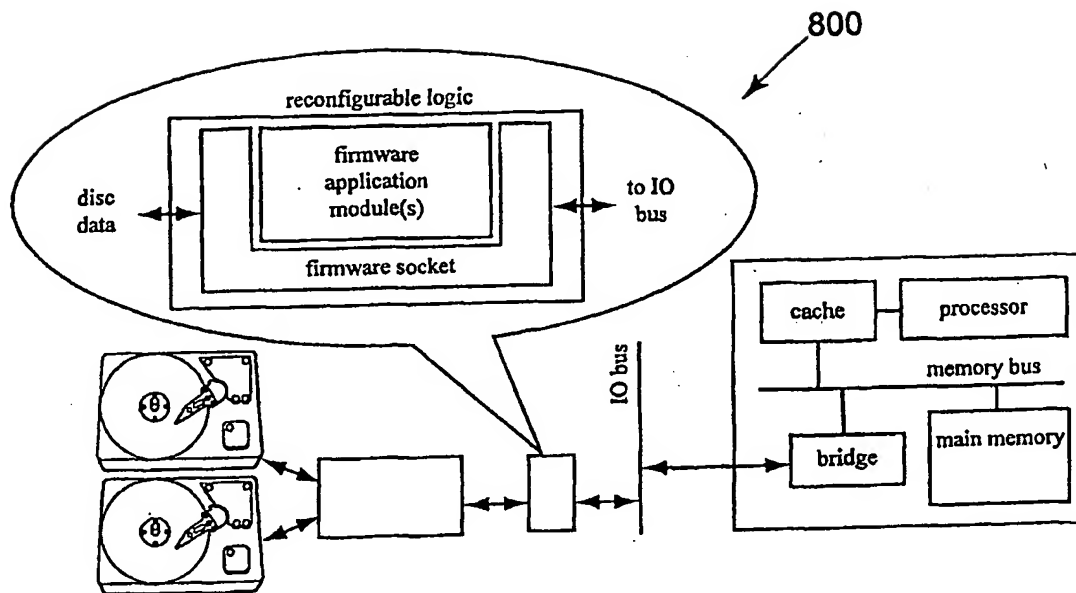


Figure 3

4/23

Sample
biosequence data
stream 120

A	C	T	G	G	T	A	C	T	A	A	C	C	G	T	C	A	G
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Figure 4

A	C	T	G	G	T	A	C	T	A	A
---	---	---	---	---	---	---	---	---	---	---

C	T	G	G	T	A	C	T	A	A	C
---	---	---	---	---	---	---	---	---	---	---

T	G	G	T	A	C	T	A	A	C	C
---	---	---	---	---	---	---	---	---	---	---

G	G	T	A	C	T	A	A	C	C	G
---	---	---	---	---	---	---	---	---	---	---

G	T	A	C	T	A	A	C	C	G	T
---	---	---	---	---	---	---	---	---	---	---

T	A	C	T	A	A	C	C	G	T	C
---	---	---	---	---	---	---	---	---	---	---

A	C	T	A	A	C	C	G	T	C	A
---	---	---	---	---	---	---	---	---	---	---

C	T	A	A	C	C	G	T	C	A	G
---	---	---	---	---	---	---	---	---	---	---

Figure 5

Sample w-mers
produced by Input
Processing Unit
104 from stream
120 and to the
Bloom Filter
Stage 106 (where
 $w=11$ and where
 $q=8$)

5/23

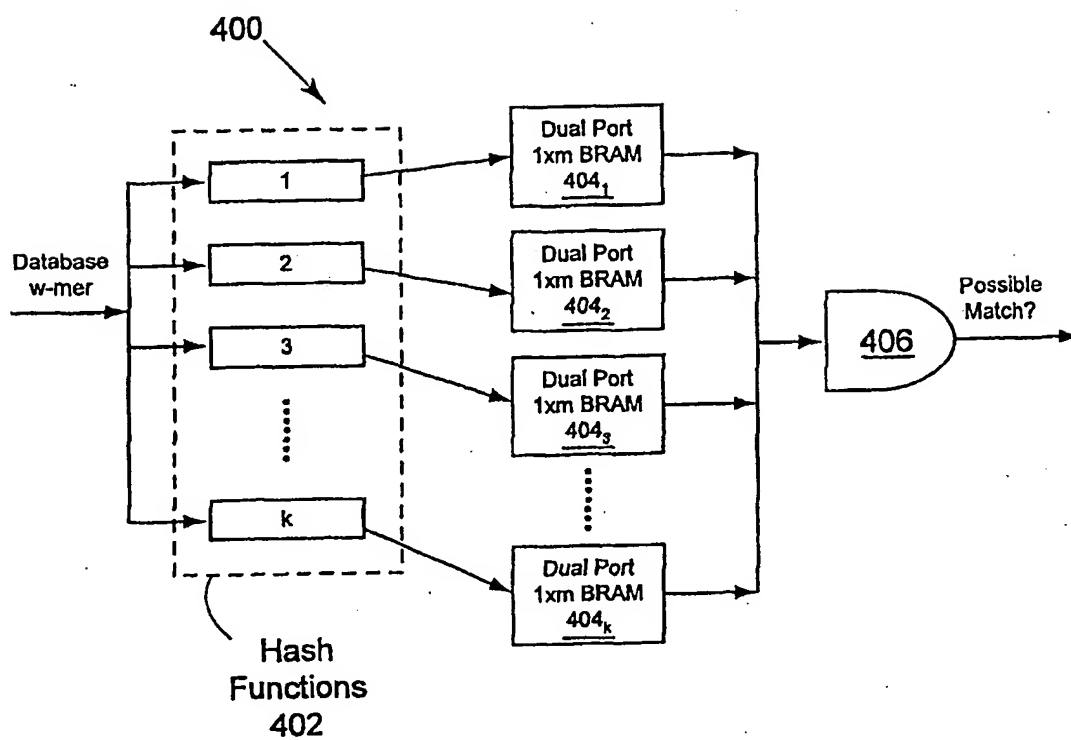


Figure 6(a)

6/23

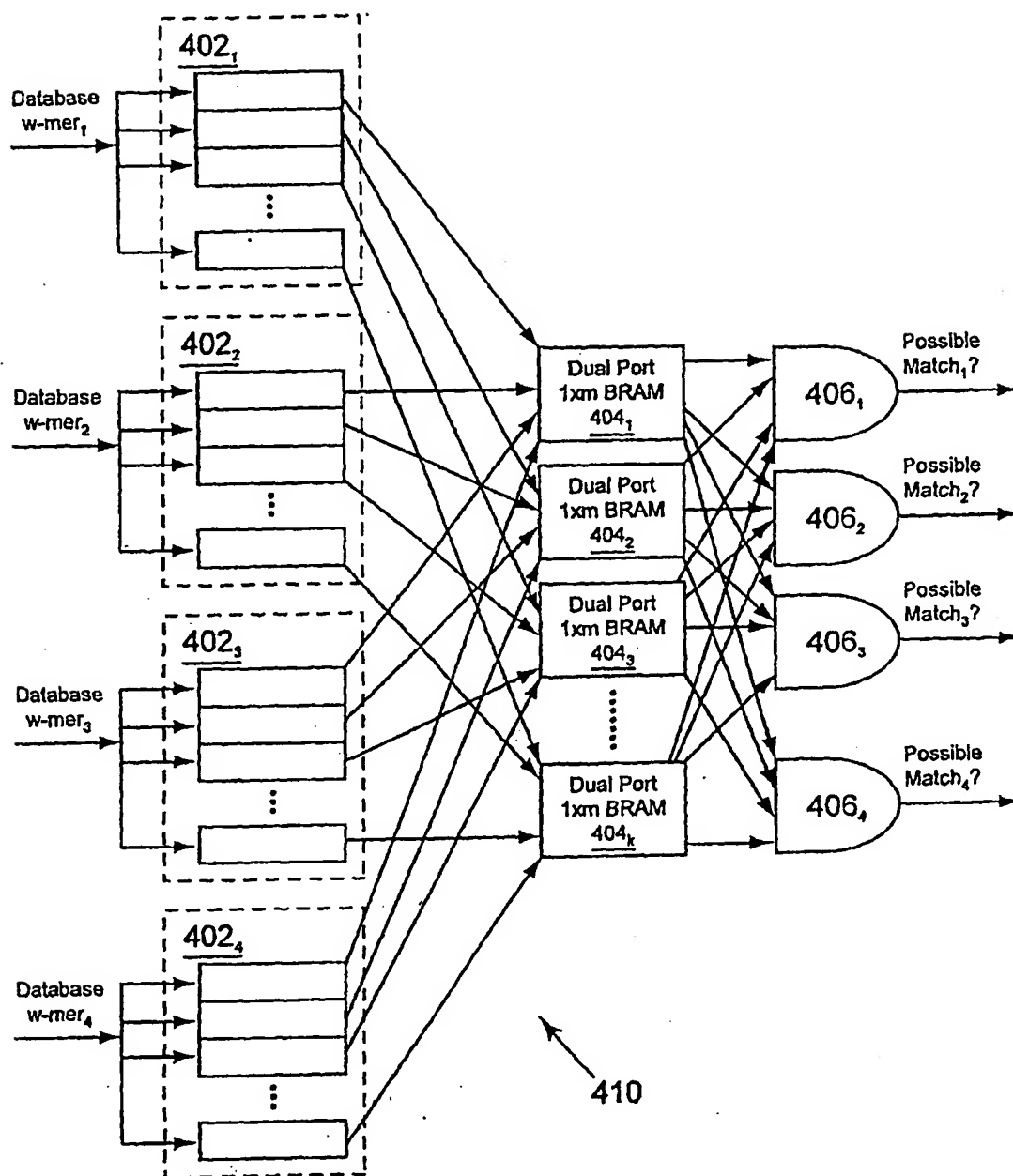


Figure 6(b)

7/23

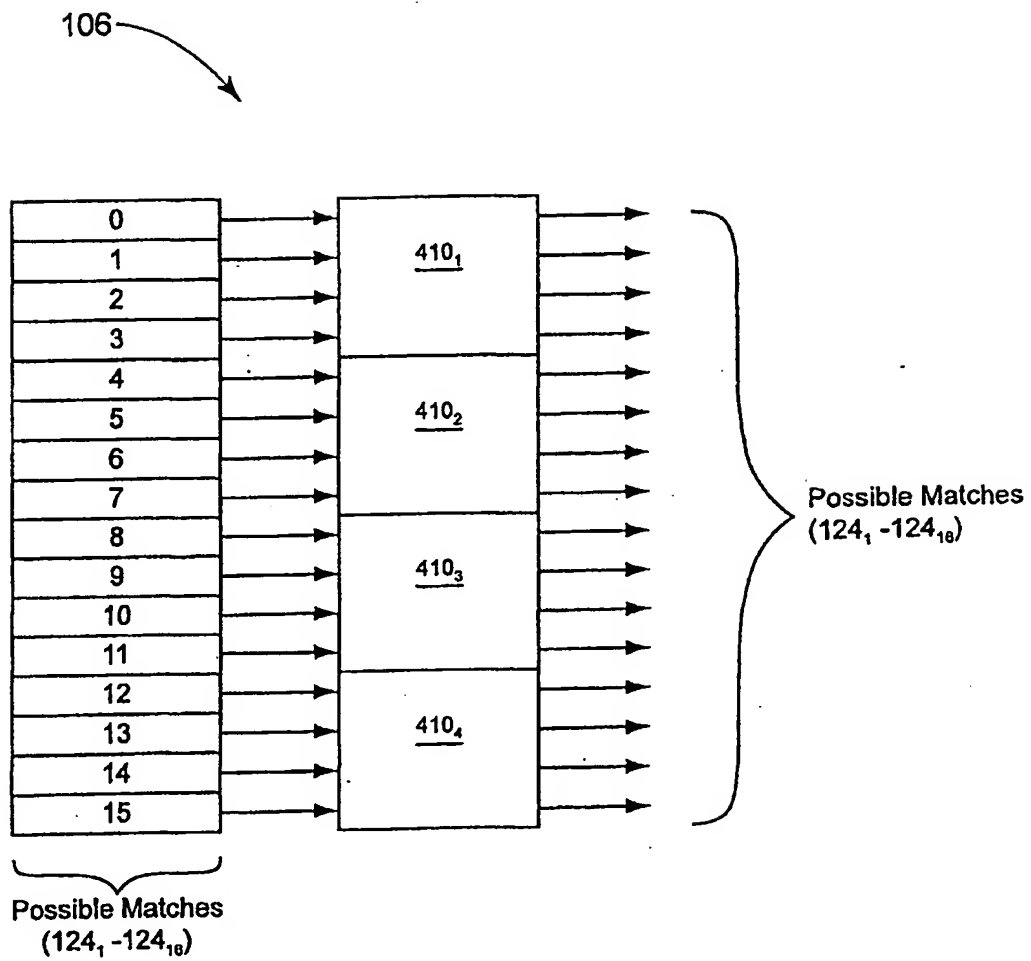


Figure 6(c)

8/23

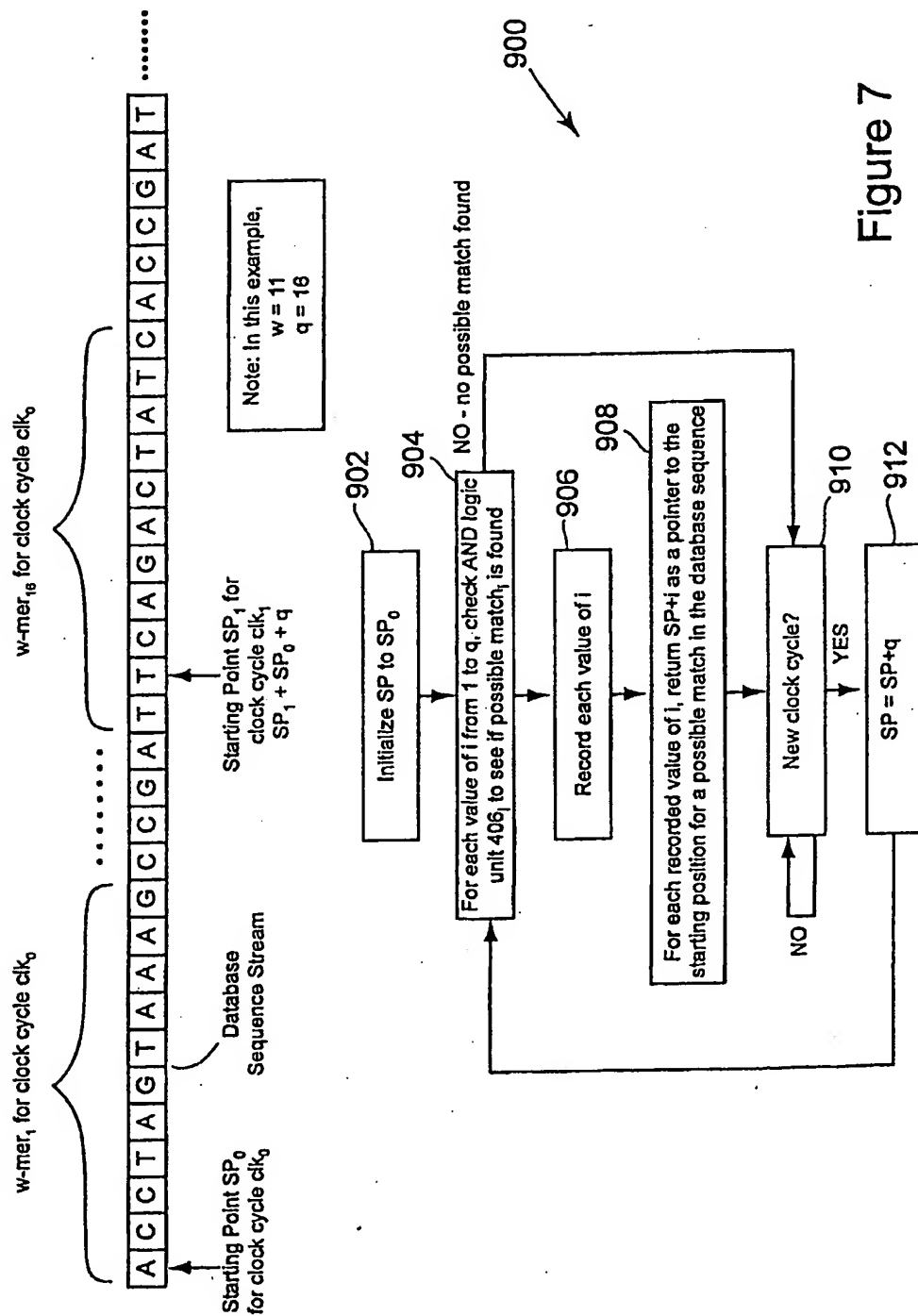


Figure 7

9/23

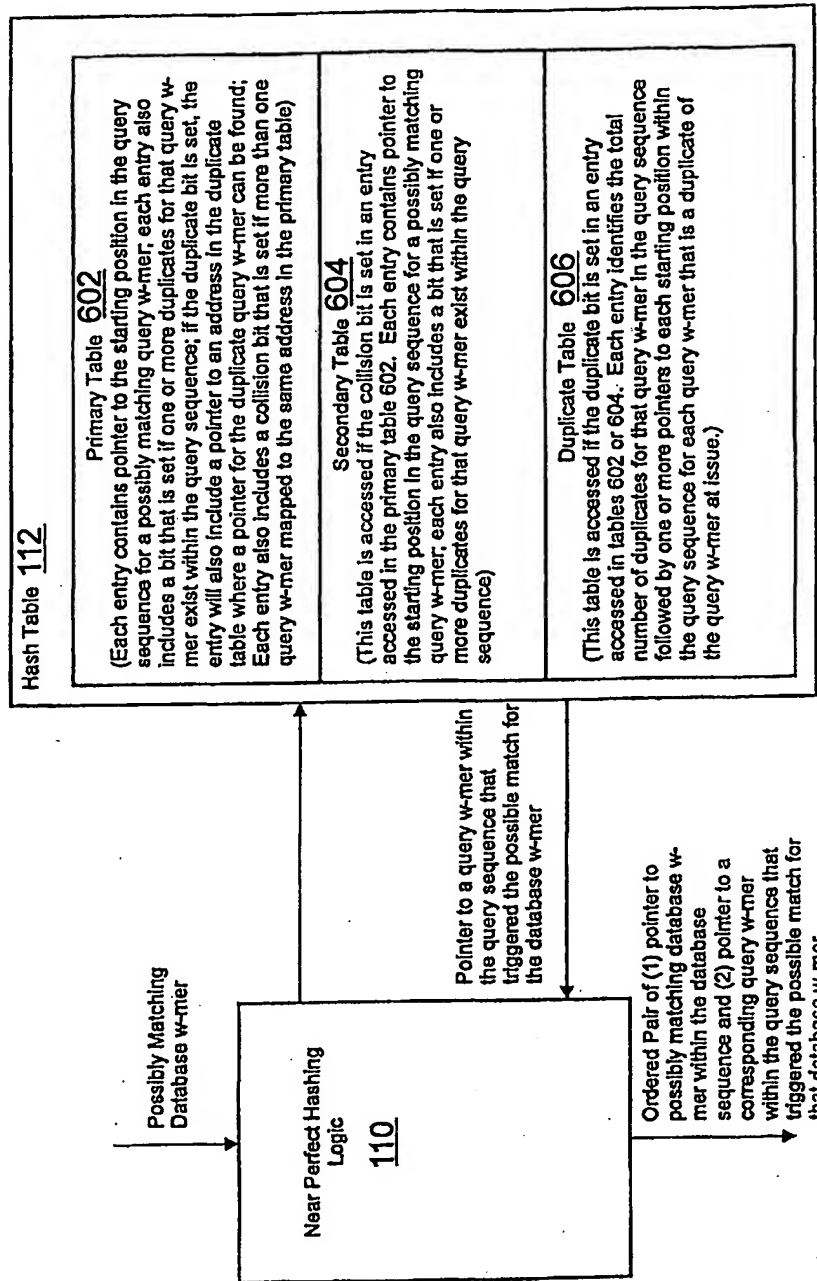
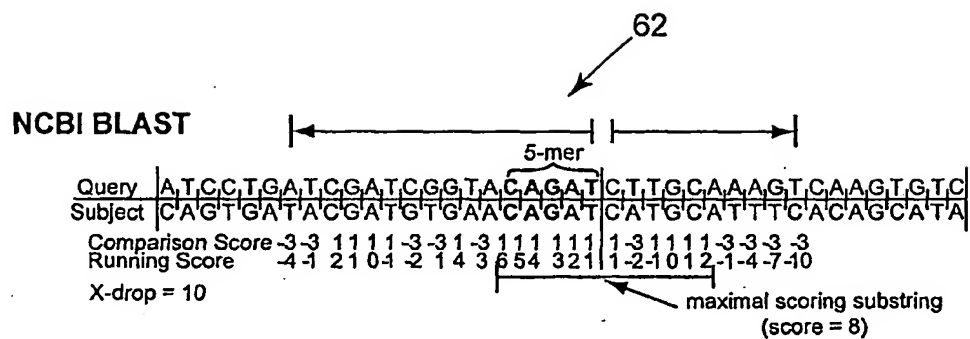


Figure 8

10/23



PRIOR ART

Figure 9

11/23

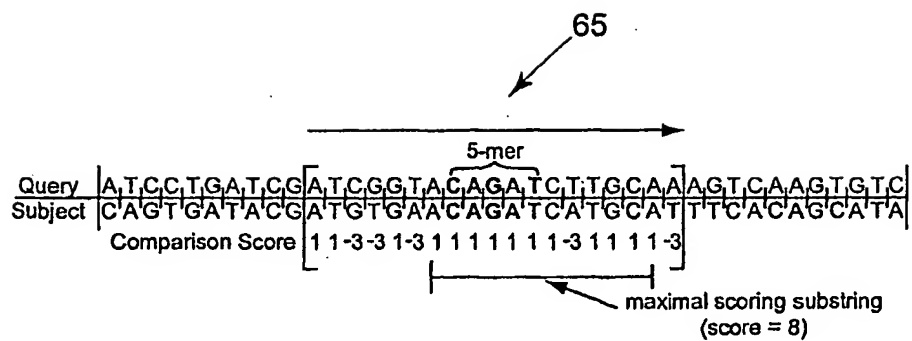


Figure 10

12/23

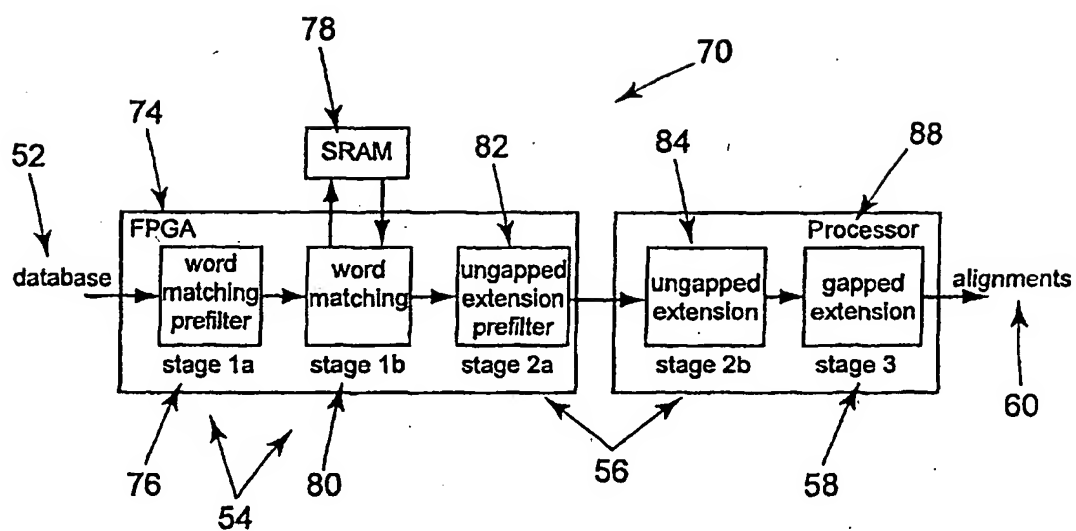


Figure 11

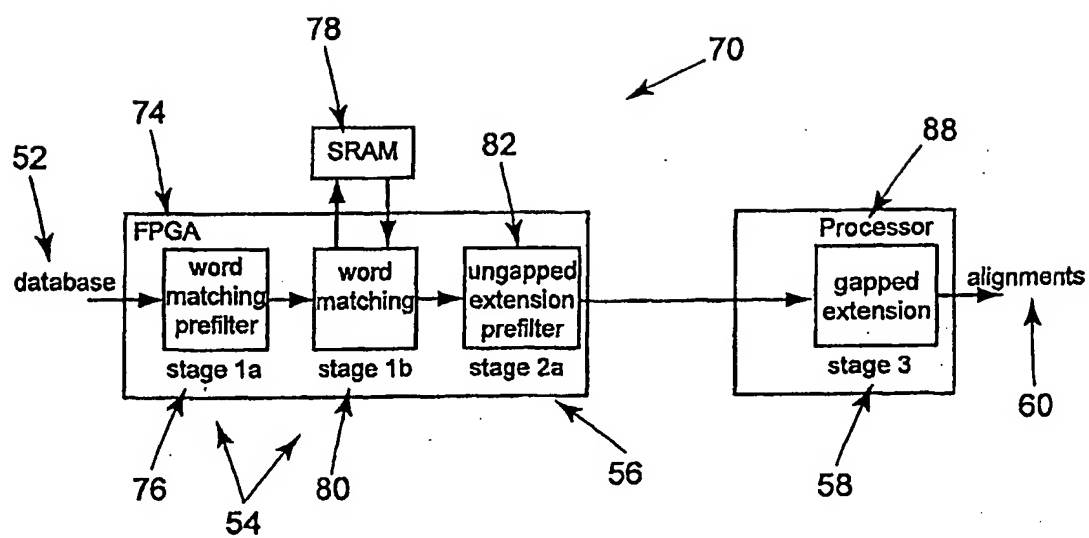


Figure 11A

14/23

66
↙

```
1  Extension (w-mer)
2  Calculate window boundaries
3   $\Gamma = \gamma = 0$ 
4   $B = B_{max} = E_{max} = 0$ 
5
6  for  $i = 1 \dots Lw$ 
7      if  $q_i = s_i$ 
8           $\gamma = \gamma + \alpha$ 
9      else
10          $\gamma = \gamma - \beta$ 
11
12     if  $\gamma > 0$ 
13         if  $\gamma > \Gamma$  and  $i > W_{merEnd}$ 
14              $\Gamma = \gamma$ 
15              $B_{max} = B$ 
16              $E_{max} = i$ 
17         else if  $i < W_{merStart}$ 
18              $B = i + 1$ 
19              $\gamma = 0$ 
20
21     if  $\Gamma > T$  or  $B_{max} = 0$  or  $E_{max} = Lw$ 
22         return True
23     else
24         return False
```

Figure 12

15/23

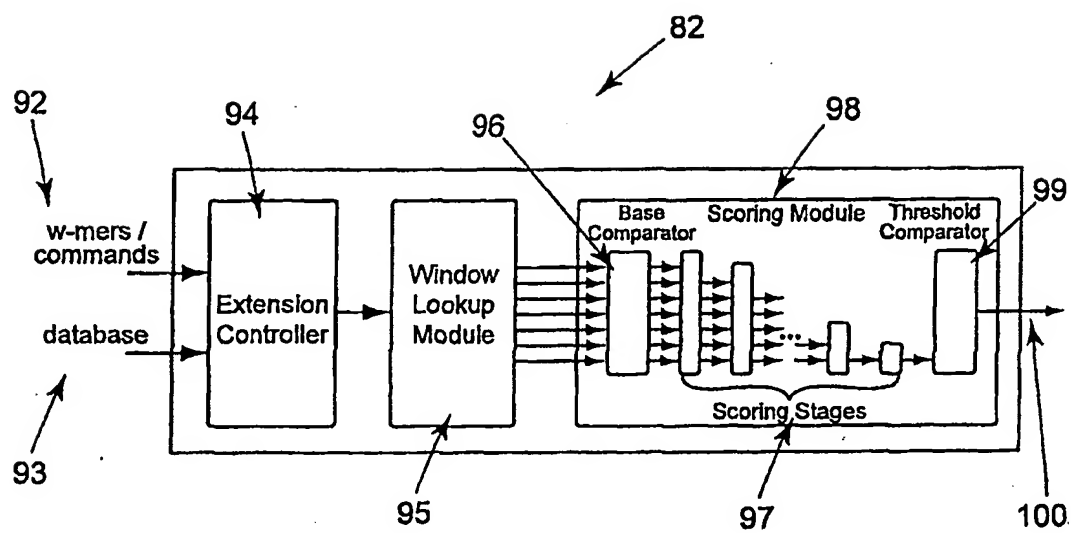


Figure 13

16/23

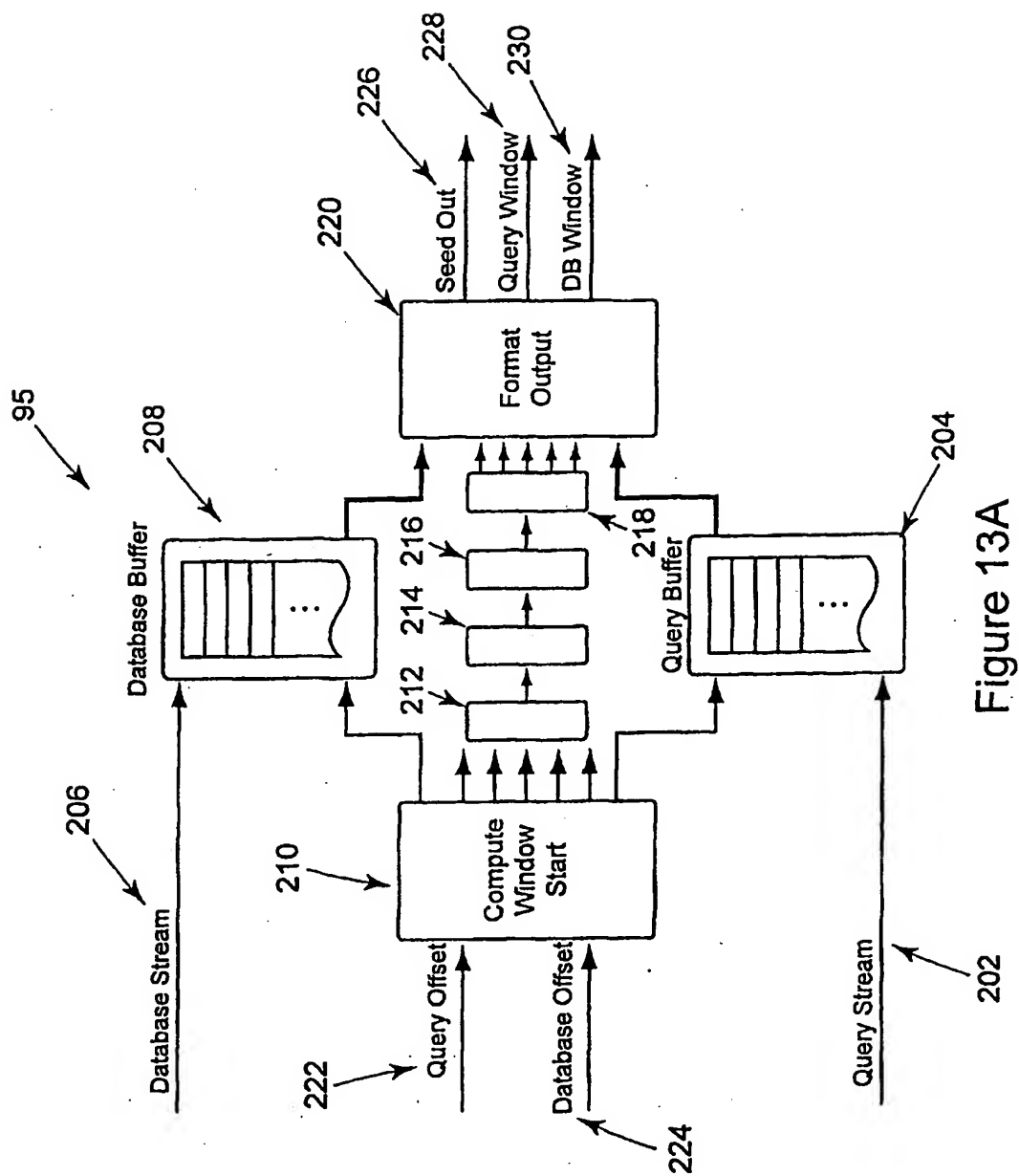


Figure 13A

17/23

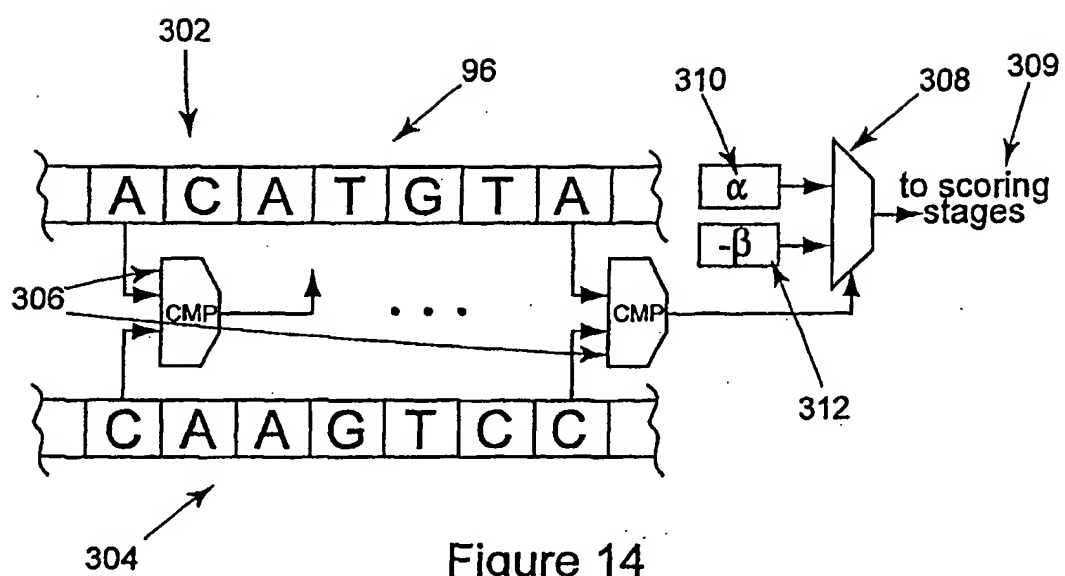


Figure 14

18/23

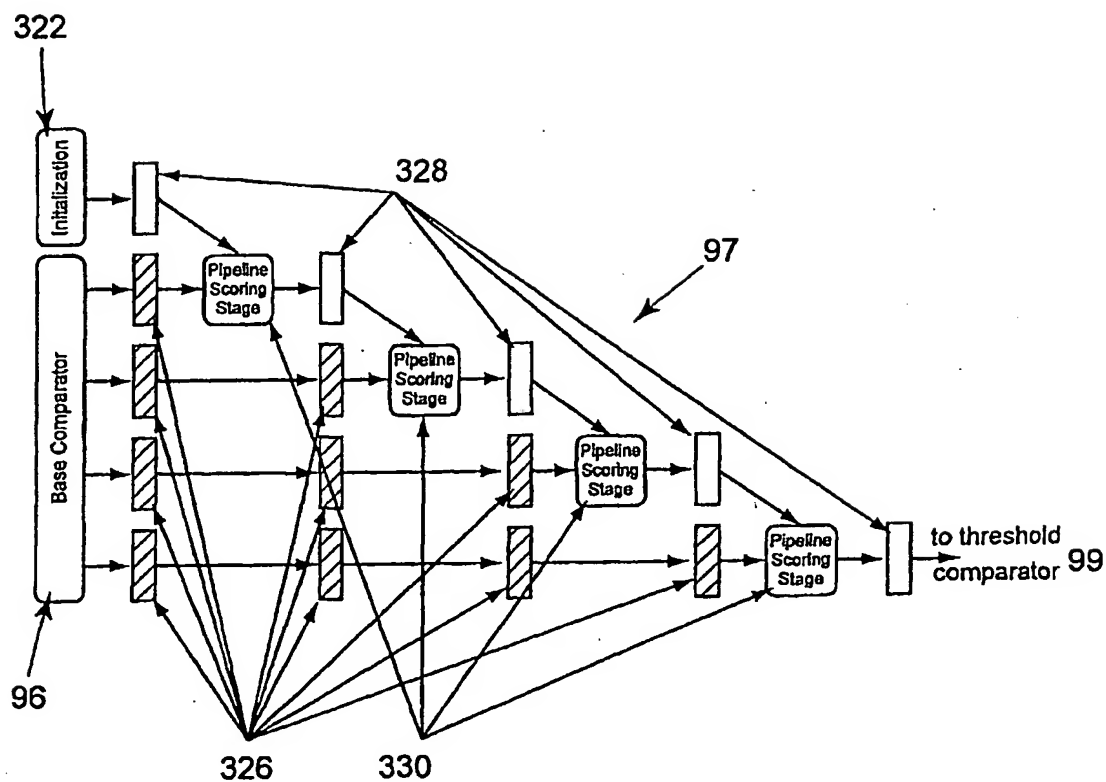


Figure 15

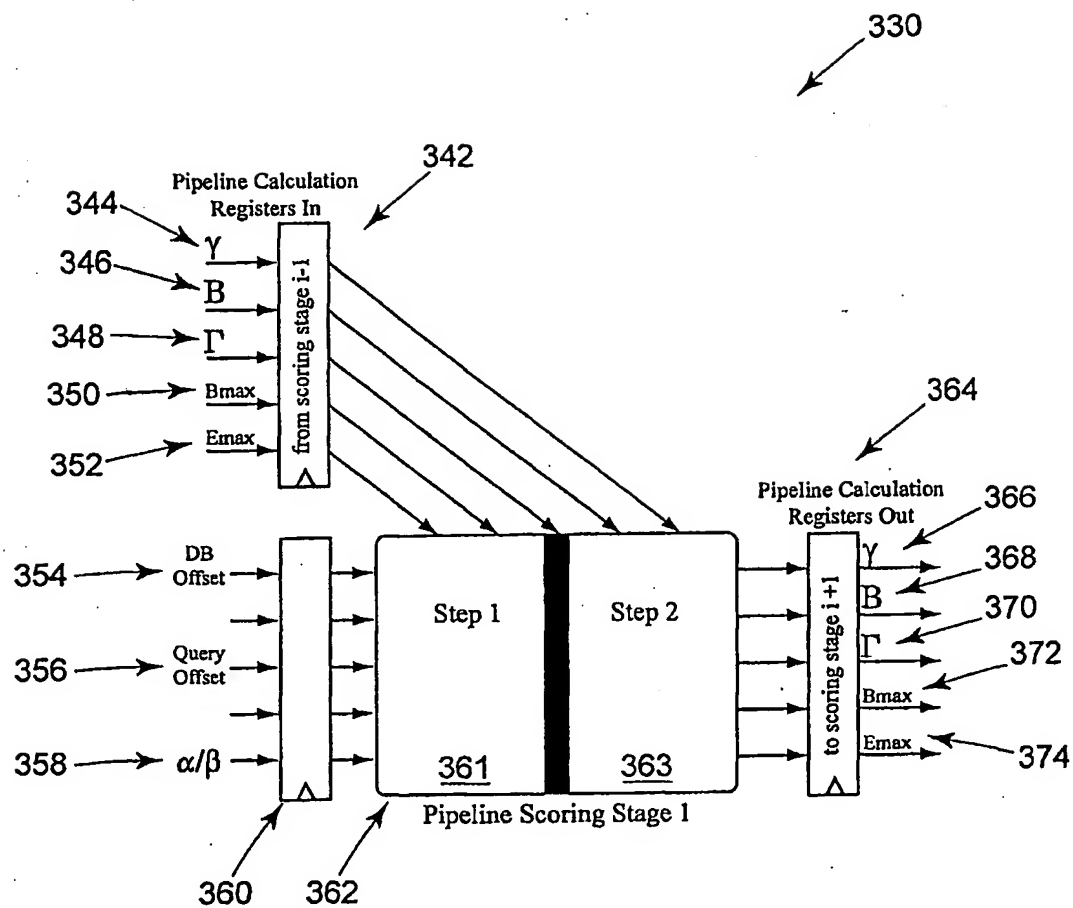


Figure 16

20/23

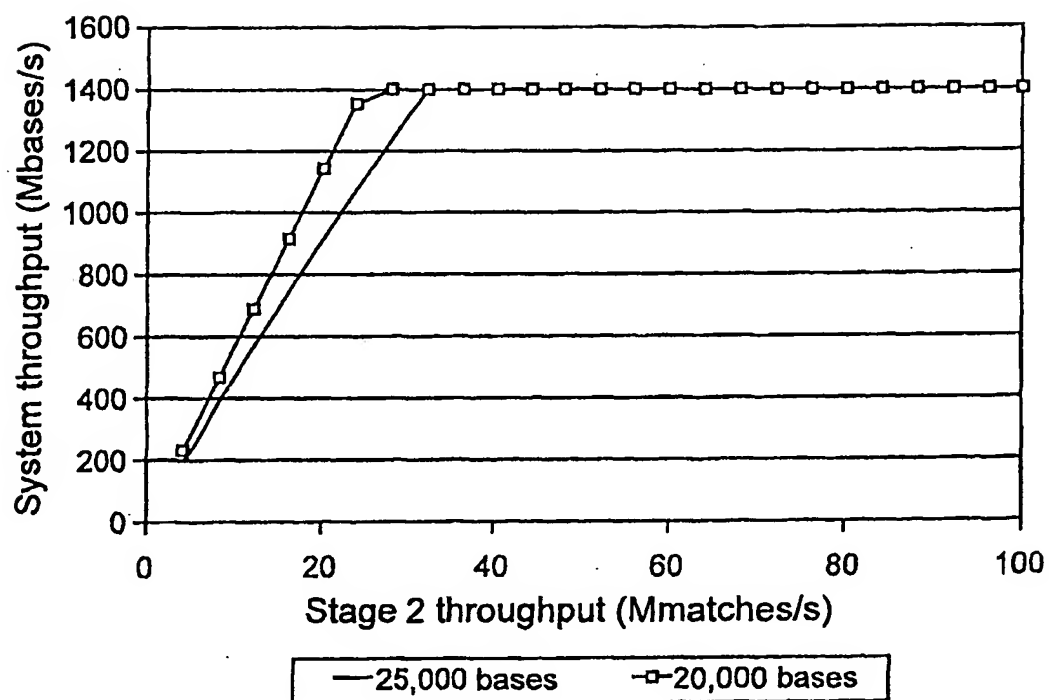


Figure 17

21/23

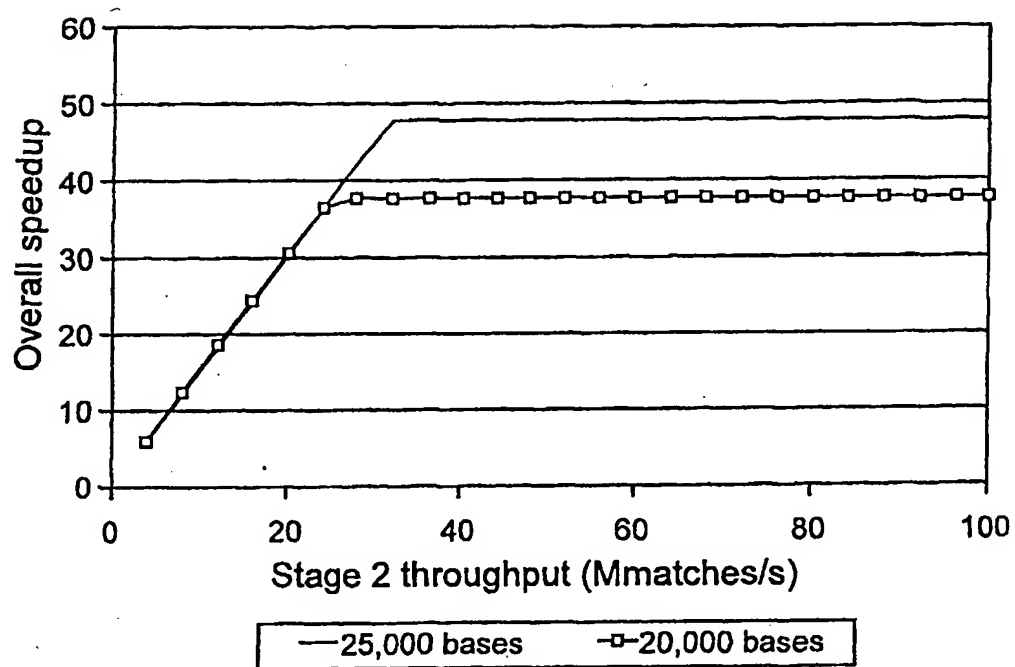



Figure 18

22/23

380



Score Threshold	Reject Fraction	Percent Found
20	0.999902	99.72
18	0.999506	99.92
17	0.997221	99.95
16	0.995735	99.96

Figure 19

23/23

Figure 20

